

# EFFICIENT TRAINING AND COMPRESSION OF DEEP NEURAL NETWORKS

A Study of Compressed Models for Text and Images



JAMES O' NEILL

Doctor of Philosophy  
Department of Computer Science  
Science and Engineering  
University of Liverpool

November 2021



Dedicated to my family and friends for their support over the past few  
years.



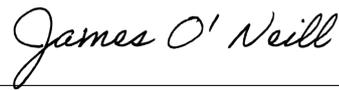
---

DECLARATION

---

I hereby declare that, this thesis (and the work presented in it) is submitted to the Department of Computer Science, University of Liverpool, in fulfilment of the requirements for the degree of Doctor of Philosophy, and that this thesis is entirely my own work.

*Liverpool, November 2021*



---

James O' Neill



*The three great essentials to achieve anything worthwhile are first, hard work; second, stick-to-itiveness; third, common sense.*

— Thomas Edison

---

## ACKNOWLEDGMENTS

---

This thesis marks the end of a three and a half year journey at Liverpool, a very memorable and gratifying period of my life. I have grown both personally and professionally throughout this time and have gained experience in both academic and industrial settings. This would not have been possible without the support of many people.

I would like to begin by thanking my family for their constant support and communication, none of this would be possible without them.

I would also like to thank my supervisor Prof. Danushka Bollegala for supporting me and providing feedback and guidance throughout my PhD. I would like to thank Dr. Angrosh Mandya, Dr. Huda Hakami, Michael Abaho, Dr. Xia Cui, Yi Zhou and Dr. Mohammad Alsuihaibani of the Natural Language Processing (NLP@Liv) Group at the Computer Science Department. I learned during our weekly paper reviews and general discussions outside of meetings.

I would also like to thank my secondary supervisor Prof. Katie Atkinson and my IPAP member Prof. Frans Coenen and Dr. Shan Luo for their valuable feedback on yearly review meetings.

I also owe a thanks to collaborators Dr. Aram Galstyan and Dr. Greg Ver Steeg for providing me the opportunity to intern at the Information Sciences Institute (ISI) that is part of the University of Southern California (USC) in Los Angeles. Working in California for 4 months was unforgettable and I hope to return once again after the pandemic.

I thank the NLP Amazon Search team in Tokyo for the experience I gained during my two internships. I gained further experience in applying machine learning in the industrial setting, but also working specifically with AWS technologies to deploy models in production to improve retrieval in one of the largest search systems in the world. I would specifically like to thank Tetsuaki Otsuki, Johannes Goller, Ryuichi Kiryo, Kosuke Tsujino, Jia Shen Boon, Jie Zhou, Motoko Kubota, Haruki Yukawa and Dmitri Lapnik. Not only for the technical experience, but for introducing me to Japanese culture, the way of life and for providing me the opportunity to live in one of the most unique cities I have ever been to.

I thank the team members I most recently worked alongside during Huawei, close to my hometown, in Dublin, Ireland. For the past six months it has been a pleasure learning from the team and providing my input on my internship and PhD topic, model compression. I look forward to returning upon the submission of this thesis.

Lastly, my PhD would not have been possible without the funding for my study at the University of Liverpool. I extend my thanks to the Department of Computer Science at the University of Liverpool, which equipped me with everything I needed to carry out my research.

---

## ABSTRACT

---

The application of deep neural networks is widespread throughout the world and is responsible for many crucial applications such as self-driving cars, machine translation, spoken language recognition, procedural content generation and medical diagnosis to name a few. However, improving the performance of neural networks generally corresponds to deeper and wider architectures, increasing the parameter count, training time and inference time to scales that exceed the typical resources available to the majority of machine learning practitioners. Neural network compression is an area of research that can address these concerns by reducing the size the network while aiming to maintain the performance of the network prior to compression. Although this research area has been somewhat active for the past three decades, it has seen a notable and proportional resurgence recently due to the rate of model size increase in deep neural networks. In this context, there are still various limitations to current compression methods and their applicability to current neural networks used for natural language processing and computer vision, which this thesis aims to address.

Firstly, many compression methods sparsify networks which leads to a theoretical parameter reduction but practically this does not lead to a reduction in storage and inference time because current hardware is not designed to implement sparse matrix multiplications efficiently. Therefore, in practice, dense matrix multiplications are carried out on a sparse network by multiplying the parameter tensors with a binary mask, leading to *more* parameters, not less. Dynamic weight sharing techniques have been under-explored as an alternative to structured pruning techniques that aim to avoid this pragmatic challenge related to efficiently using sparse networks post-compression. Hence, in this thesis we discuss dynamic weight sharing techniques that aim to preserve density in the network without zeroing out whole structures.

Secondly, compression methods are typically evaluated in the supervised learning setting. Thus, little is known about how our assumptions in the supervised learning setting hold against other settings such as few-shot transfer learning or zero-shot domain adaptation (e.g zero-shot cross-lingual transfer when using cross-lingual models). Therefore, we explore how iterative pruning behaves in the few-shot and zero-shot cases.

Thirdly, compression methods such as pruning and knowledge distillation have primarily been adopted in isolation, without much insight as to how they might be used in tandem to further boost compression performance. We also investigate whether this is viable and how both

can be used simultaneously to reduce the requirement of a two-stage compression process.

Lastly, compression is usually carried out on the classification model. However, in natural language processing we often learn an embedding model that outputs character, sub-word or word representations that are used as input to the classifier. Hence, we also explore compression methods that reconstruct an ensemble set of sub-word and word representations, where the resulting learned *meta-embeddings* are used as input to classification models for downstream tasks, generally outperforming the defacto single sub-word or word representations that are typically used.

Hence, this thesis investigates and proposes novel compression methods and more efficient training of pretrained deep networks that improve the current state of the art in domains such as natural language and computer vision. This broadly includes contributions to knowledge distillation, pruning, dynamic weight sharing and improving fine-tuning in the transfer learning setting.

# Contents

1	INTRODUCTION	1
1.1	Motivation	2
1.2	Research Aims	4
1.3	Research Questions	7
1.4	Contributions	8
1.5	Publications and Code	12
1.6	Thesis Outline and Summary	13
2	BACKGROUND AND RELATED WORK	15
2.1	Weight Sharing	16
2.1.1	Clustering-based Weight Sharing	16
2.1.2	Learning Weight Sharing	17
2.1.3	Weight Sharing in Large Architectures	18
2.1.4	Reusing Layers Recursively	19
2.2	Network Pruning	20
2.2.1	Categorizing Pruning Techniques	21
2.2.2	Pruning using Weight Regularization	23
2.2.3	Pruning via Loss Sensitivity	24
2.2.4	Structured Pruning	30
2.2.5	Search-based Pruning	35
2.2.6	Pruning Before Training	39
2.3	Low Rank Matrix & Tensor Decompositions	42
2.3.1	Tensor Decomposition	43
2.3.2	Tensor Decomposition For Attention and Recurrent Layers	44
2.3.3	Applications of Tensor Decompositions to Convolutional Layers	45
2.4	Knowledge Distillation	46
2.4.1	Analysis of Knowledge Distillation	47
2.4.2	Data-Free Knowledge Distillation	50
2.4.3	Distilling Recurrent (Autoregressive) Neural Networks	51
2.4.4	Distilling Transformer-based (Non-Autoregressive) Networks	51
2.4.5	Ensemble-based Knowledge Distillation	53
2.4.6	Meta-Embedding Problem Definition	56
2.4.7	Reinforcement Learning Based Knowledge Distillation	63
2.4.8	Generative Modelling Based Knowledge Distillation	64
2.4.9	Pairwise-based Knowledge Distillation	66

2.5	Quantization . . . . .	69
2.5.1	Approximating High Resolution Computation . .	70
2.5.2	Adaptive Ranges and Clipping . . . . .	71
2.5.3	Robustness to Quantization and Related Distortions	72
2.5.4	Retraining Quantized Networks . . . . .	72
2.6	Summary . . . . .	79
3	MITIGATING EXPOSURE BIAS	81
3.1	Introduction . . . . .	81
3.2	Related Work . . . . .	82
3.3	Methodology . . . . .	84
3.3.1	Addressing Compounding Errors . . . . .	85
3.3.2	Experimental Results . . . . .	90
3.3.3	Evaluation . . . . .	93
3.3.4	Summary . . . . .	95
4	MODEL COMPRESSION VIA ERROR-CORRECTING CODES	97
4.1	Introduction . . . . .	97
4.2	Background . . . . .	99
4.3	Methodology . . . . .	101
4.3.1	Codebook Construction . . . . .	101
4.3.2	Latent Variable Mixture Sampling . . . . .	103
4.3.3	Differentiable Latent Variable Sampling . . . . .	104
4.4	Experimental Setup . . . . .	106
4.5	Results . . . . .	107
4.6	Summary . . . . .	110
5	MODEL COMPRESSION VIA META-EMBEDDING	111
5.1	Introduction . . . . .	111
5.1.1	Multi-Task Learning Representations . . . . .	114
5.2	Methodology . . . . .	115
5.3	Autoencoders for Meta-Embedding . . . . .	115
5.3.1	Model Configurations . . . . .	116
5.3.2	Target Autoencoder Meta-Embedding . . . . .	117
5.4	Meta-Embedding for Supervised Task Regularization . .	117
5.4.1	Multi-Task Learning for Intrinsic Tasks . . . . .	117
5.4.2	Multi-Task Learning for Extrinsic Tasks . . . . .	120
5.5	Experiments . . . . .	121
5.5.1	Intrinsic Evaluation . . . . .	121
5.5.2	Extrinsic Evaluation . . . . .	125
5.5.3	Downstream Task Results . . . . .	126
5.6	Summary . . . . .	127
6	KNOWLEDGE DISTILLED REINFORCEMENT LEARNING	129
6.1	Introduction . . . . .	129
6.2	Methodology . . . . .	131
6.2.1	Image Caption Setup . . . . .	131
6.2.2	Policy Gradient Training . . . . .	132
6.2.3	Transfer Reward Learner . . . . .	133
6.3	Experimental Setup . . . . .	135

6.3.1	Dataset Details . . . . .	135
6.3.2	Training Details . . . . .	136
6.3.3	Embedding Similarity Evaluation . . . . .	136
6.4	Results . . . . .	137
6.4.1	Flickr30k . . . . .	137
6.4.2	MSCOCO . . . . .	138
6.5	Summary . . . . .	140
7	KNOWLEDGE DISTILLED METRIC LEARNING	141
7.1	Introduction . . . . .	141
7.2	Related Research . . . . .	143
7.3	Methodology . . . . .	144
7.3.1	Conditioned Negative Sampling . . . . .	145
7.3.2	Interpolating Contrastive Representations . . . . .	149
7.3.3	Theoretical Analysis of Conditioned Sampling . . . . .	149
7.4	Experiments . . . . .	152
7.4.1	Summary . . . . .	156
8	MODEL COMPRESSION VIA LAYER FUSION	159
8.1	Introduction . . . . .	159
8.2	Related Work . . . . .	161
8.3	Methodology . . . . .	162
8.3.1	Desirable Properties of Weight Similarity . . . . .	163
8.3.2	Layer Alignment & Layer Similarity . . . . .	163
8.3.3	Fusing Layers . . . . .	165
8.4	Experimental Details . . . . .	166
8.4.1	Compression Without Retraining . . . . .	167
8.4.2	Layer Fusion & Compression ReTraining . . . . .	168
8.5	Results . . . . .	168
8.5.1	Image Classification . . . . .	168
8.5.2	Language Modeling . . . . .	170
8.6	Summary . . . . .	173
9	ITERATIVE PRUNING IN THE ZERO-SHOT SETTING	175
9.1	Introduction . . . . .	175
9.2	Related Work . . . . .	177
9.3	Methodology . . . . .	178
9.3.1	<i>AlignReg</i> Between Pruned and Unpruned Layers . . . . .	179
9.3.2	Connections to Knowledge Distillation . . . . .	181
9.4	Experimental Setup . . . . .	181
9.5	Results . . . . .	183
9.6	Summary . . . . .	186
10	GRADIENT SPARSIFICATION FOR EFFICIENT FINE-TUNING OF TRANSFORMERS	189
10.1	Introduction . . . . .	189
10.2	Related Research . . . . .	190
10.2.1	Language Model Fine-Tuning . . . . .	190
10.2.2	Language Model Fine-Masking . . . . .	192
10.3	Methodology . . . . .	192

10.3.1	Gradient Dropout . . . . .	193
10.3.2	Epoch-wise Gradient Dropout . . . . .	193
10.3.3	Annealed Variants of Gradient Dropout . . . . .	194
10.3.4	Experimental Details . . . . .	194
10.4	Results . . . . .	196
10.4.1	Sentence Classification Results . . . . .	196
10.4.2	Pairwise Classification . . . . .	197
10.4.3	Structured Prediction Tasks . . . . .	199
10.4.4	Sentence and Span Retrieval Tasks . . . . .	200
10.4.5	An Analysis of Training Stability . . . . .	201
10.4.6	XGLUE Understanding Score . . . . .	201
10.4.7	Per Language Analysis . . . . .	202
10.5	Summary . . . . .	203
11	SELF-DISTILLED PRUNING . . . . .	205
11.1	Introduction . . . . .	205
11.2	Background and Related Work . . . . .	207
11.3	Proposed Methodology . . . . .	208
11.3.1	Maximizing Cross-Correlation Between Pruned and Unpruned Embeddings . . . . .	208
11.3.2	A Frobenius Distortion Perspective of Self-Distilled Pruning . . . . .	210
11.3.3	How Does Self-Distillation Improve Pruned Model Generalization ? . . . . .	211
11.4	Experimental Setup . . . . .	213
11.5	Empirical Results . . . . .	214
11.6	Summary . . . . .	220
12	CONCLUSION . . . . .	221
12.1	Thesis Summary . . . . .	221
12.2	Contributions and Findings . . . . .	225
12.3	Future Work . . . . .	228
12.3.1	Unify Compression Approaches . . . . .	228
12.3.2	More Compression Work on Large Non-Sparse Architectures . . . . .	228
12.3.3	Automatically Choosing Student Size in Model Distillation . . . . .	229
12.3.4	Bridging the Gap Between Data and Model Com- pression . . . . .	229
12.3.5	Further Theoretical Analysis . . . . .	229
<b>I APPENDIX</b>		
	BIBLIOGRAPHY . . . . .	233
A	APPENDIX . . . . .	279
A.1	Further Details on Layer Fusion . . . . .	279
A.2	Conditional Negative Sampling . . . . .	280
A.2.1	Hardware Details . . . . .	280

A.2.2	Network Architectures . . . . .	280
A.2.3	Experiment Details . . . . .	280
A.2.4	Metric Learning Distillation Objectives . . . . .	283
A.2.5	Connection Between Mutual Information & Con- ditional Negative Sampling . . . . .	284
A.2.6	Uniform vs SCNS Sample Complexity . . . . .	287
A.2.7	Additional Results . . . . .	290
A.3	Layer Fusion . . . . .	291
A.3.1	Layer Fusion & Compression ReTraining Details	293
A.3.2	Approximating the Covariance Matrix . . . . .	294
A.3.3	Similarity Between Covariance Measures . . . . .	295

# List of Figures

Figure 1.1	Accuracy vs Num. of Parameters for CNN architectures (source on left: Tan and Le [408]) and Num. of Parameters vs Years for Transformers (source on right: Sanh [372]) . . . . .	2
Figure 1.2	Thesis Diagram. Green indicates chapters focused on efficient training, blue is for chapters on model compression and gradients of both indicates chapters which pertain to both efficient training and model compression. . . . .	13
Figure 3.1	Perplexity on WikiText2: SS and NNRS . . . . .	93
Figure 4.1	Curriculum Mixture Sampling . . . . .	103
Figure 4.2	ECOC-NSP Peplexity vs. Decoder Parameters (corresponding to 14/20/40 codeword bits for Penn-TreeBank and 17/40/100 codeword bits for WikiText-2/103) . . . . .	107
Figure 4.3	WikiText-2 Validation Perplexity When Varying $\tau$ (corresponding dashed lines) in CLVMS-ECOC (LSTM) . . . . .	109
Figure 5.1	Word Association Annotation Distribution . . . . .	118
Figure 5.2	Multi-Task Meta-Embedding (an example of <i>true</i> similarity). An illustration of how the meta-embeddings of cup and tea are used as input to the proposed architecture for word similarity tasks, where the meta-embedding hidden layer is shared for reconstruction and word similarity prediction. . . . .	119
Figure 5.3	LSTM Results with Multi-Task Meta-Embedding on Universal Dependency PoS Tagging . . . . .	126
Figure 6.1	Actor $\pi_\theta(\cdot)$ produces actions $\hat{Y}$ given an encoded image $h_s$ and caption $X_t$ that are passed to $\text{TRL}(Y, \hat{Y})$ and encoded into $(h_1, h_2)$ respectively and scored with action-value $A^\pi = V_\Theta^\pi(s) - Q^\pi(s, a)$ . . . . .	133
Figure 6.2	Qualitative Results on MSCOCO . . . . .	139
Figure 7.1	CIFAR-100 subset of word embedding class similarities . . . . .	146
Figure 7.2	CIFAR-100 Test Accuracy for Knowledge Distillation Approaches . . . . .	152

Figure 7.3	Effect of $\beta$ in Latent Mixup and <i>kin</i> SCNS . . .	154
Figure 7.4	CIFAR-100 Convergence Time Comparison . . .	155
Figure 8.1	CIFAR-10 Test Accuracy With And Without Retraining . . . . .	168
Figure 8.2	Wikitext-2: Pruning Without Retraining . . . .	172
Figure 8.3	Euclidean Distance Between Trans-XL Weights	173
Figure 8.4	Language Modeling Compression on WikiText-2 with Retraining . . . . .	173
Figure 9.1	English and Zero-Shot Test Accuracy on News Classification. . . . .	181
Figure 9.2	Zero-Shot Test F1 on Named Entity Recognition.	183
Figure 9.3	Test Accuracy on Question Answer Matching. .	183
Figure 9.4	Part of Speech Tagging Test Accuracy. . . . .	184
Figure 9.5	Zero-Shot XNLI Results Per Language After Iteratively Fine-Pruning XLM-RoBERTa <sub>Base</sub> . .	187
Figure 9.6	Weight Distribution and Weight Norms Per Layer of Remaining Weights After Iterative Pruning .	188
Figure 9.7	Gradient Distribution and Gradient Norms Per Layer of Remaining Weights After Iterative Pruning . . . . .	188
Figure 9.8	Class Separability Between Class Representations At Each Iterative Pruning Step on PAWSX.	188
Figure 10.1	GradDrop-Epoch-Toggle (Top) and GradDrop-Epoch (Bottom). Grey represents frozen gradients, blue represents active gradients where darker blue indicates the recency of gradients turned on. . . . .	194
Figure 10.2	Test Performance Per Training Epoch. . . . .	202
Figure 10.3	Test Performance Increase By Language in POS and XNLI. Red bars indicate accuracy increases or decreases, while blue indicates the fractional increase of GradDrop over standard fine-tuning.	203
Figure 11.1	Self-Distilled Pruning w/ a Cross-Correlation Knowledge Distillation Loss. . . . .	209
Figure 11.2	<b>Iterative Pruning Test Performance on GLUE tasks.</b> . . . . .	216
Figure 11.3	<b>Zero-Shot Results After Iteratively Fine-Pruning XLM-R<sub>Base</sub> on XGLUE tasks.</b> . .	217
Figure 11.4	Mutual Information Between Unpruned and Pruned Representations (left) and Signal-To-Noise Ratio (right) from PAWS-X Development Set Representations. . . . .	218
Figure 11.5	PAWS-X: Self-Distilled Pruning Leads to Better Performance Recovery. . . . .	219

Figure A.1	$k$ -NN Conditional Negative Sampling for embeddings on the $L_2$ sphere retrieved from the lookup table. . . . .	286
Figure A.2	Tiny-ImageNet Boxplot Test Accuracy for Knowledge Distillation Approaches . . . . .	290
Figure A.3	CIFAR-100 Test Accuracy for KD Approaches .	291
Figure A.4	Tiny-Imagenet 200 Test Accuracy for KD Approaches . . . . .	292
Figure A.5	CIFAR-100 Normalized Test Accuracy for Knowledge Distillation Approaches . . . . .	292
Figure A.6	Tiny-ImageNet 200 Test Accuracy for Knowledge Distillation Approaches . . . . .	293
Figure A.7	Transformer-XL Weight Similarity via Euclidean Distance . . . . .	299
Figure A.8	BERT Weight Similarity with Euclidean Distance	299
Figure A.9	Euclidean Distance Between Trans-XL Weights: (1) Query-Key-Value Attention, (2) Output Attention, (3, 4) FC Layers . . . . .	299

## List of Tables

Table 3.1	Test Perplexity for a 2-hidden layer LSTM [165] using Transition Probability Replacement Sampling (TPRS), Scheduled Sampling (SS) and Nearest-Neighbor Replacement Sampling (NNRS) with linear, s-shaped curve and exponential sampling functions. . . . .	91
Table 3.2	Test Perplexity for a 2-hidden layer LSTM [165] using Transition Probability Sampling (TPRS), SS and Nearest-Neighbor Replacement Sampling (NNRS) with linear, s-shaped curve and exponential sampling functions. . . . .	92
Table 3.3	PTB Self-BLEU4 and Self-WMD Validation/Test scores . . . . .	92
Table 3.4	WikiText-2 Self-BLEU4 and Self-WMD Validation/Test scores . . . . .	93
Table 3.5	WikiText-2 BLEU-4 & WMD Quality Scores . .	94

Table 4.1	Perplexities for Full Softmax (SM), Sampled SM (Sample-SM), Hierarchical-SM (HSM), Adaptive-SM, Noise Contrastive Estimation (NCE) & ECOC-NSP. . . . .	108
Table 4.2	MSCOCO Test Results on BLEU (B), ROUGE-L (R-L) & METEOR (MET) Eval. Metrics . . .	110
Table 5.1	Meta-Embedding Unsupervised Results ( $\rho_s$ ) . .	122
Table 5.2	SS-MTL Embedding Analogy Transferability . .	123
Table 5.3	Multi-Task Word Embedding Learning ( $\rho_s$ ) Results on Word Similarity . . . . .	124
Table 5.4	Validation (left) & Test (right) Accuracy (%): NER, UDPoS & Sentiment Analysis (only test acc.) . . . . .	125
Table 6.1	SoTA Methods for Flickr30k . . . . .	136
Table 6.2	Flickr30k Results for ML, Actor-Critic and our proposed TRL AC Models (using a beam width of 5) . . . . .	137
Table 6.3	MSCOCO Results for ML, Actor-Critic and our proposed TRL AC Models (B=5) . . . . .	138
Table 7.1	Test accuracy (%) of student networks on CIFAR-10. . . . .	153
Table 7.2	Test accuracy (%) of student networks on Tiny-ImageNet-200 . . . . .	154
Table 7.3	An ablation of efficient NS techniques on CIFAR-100. . . . .	156
Table 8.1	CIFAR-10 Test Accuracy with WS-Based CNN Layer Fusion . . . . .	169
Table 8.2	WikiText-2 Test Perplexity Without Fine-Tuning Or Retraining. . . . .	171
Table 8.4	WikiText-2: Perplexity With <i>LayerDrop</i> & <i>Layer Fusion</i> . . . . .	174
Table 8.3	WikiText-2: Perplexity of LF-Retraining @50% reduction . . . . .	174
Table 9.1	XGLUE Iterative Pruning @ 31% Remaining Weights of XLM-R <sub>Base</sub> - Zero Shot Cross-Lingual Performance Per Task and Overall Average Score (avg). . . . .	185
Table 10.1	XNLI zero-shot accuracy (apart from ‘en’) for each language. Results of fine-tuned XLM-R from prior work [69, 107] are from the XTREME benchmark [171]. . . . .	195
Table 10.2	Fine-Tuning XLM-R <sub>Large</sub> Results on News Classification. Test Accuracy on English and Zero-Shot Results for German, Spanish and French. . . . .	197

Table 10.3	Fine-Tuning XLM-R <sub>Large</sub> Zero-Shot Results on Question Answer Matching. Test Accuracy on German, English and French. . . . .	197
Table 10.4	Fine-Tuning XLM-R <sub>Large</sub> Results on Query-Ad Matching. Zero-shot Test Accuracy on German, English and French. . . . .	198
Table 10.5	Fine-Tuning XLM-R <sub>Large</sub> Results on Cross-lingual Adversarial Paraphrase Identification. Test F1 score on English and Zero-Shot Results in German, Spanish and French. . . . .	198
Table 10.6	Fine-Tuning XLM-R <sub>Large</sub> Results on Named Entity Recognition. Test F1 score on English and Zero-Shot Results in German, Spanish and Dutch.	199
Table 10.7	Fine-Tuning XLM-R <sub>Large</sub> Results on Part of Speech Tagging. Test F1 score on English and Zero-Shot Results in 17 other languages. . . . .	199
Table 10.8	Fine-Tuning XLM-R <sub>Large</sub> Results on Web Page Ranking. Normalized DCG on German, English, Spanish, French, Italian, Portuguese and Chinese.	200
Table 10.9	Cross-Lingual Transfer Results on MLQA. F1 on Arabic, German, English, Spanish, Hindi, Vietnamese and <i>Simplified</i> Chinese. . . . .	200
Table 10.10	Zero Shot Cross-Lingual Performance Per Task and Overall Average Score (avg). . . . .	201
Table 11.1	GLUE benchmark results for pruned models @10% (or @20%) remaining weights. . . . .	215
Table 11.2	XGLUE Iterative Pruning @ 30% Remaining Weights of XLM-R <sub>base</sub> - Zero Shot Cross-Lingual Performance Per Task and Overall Average Score (Avg). . . . .	218
Table A.1	Abbreviations for Knowledge Distillation Baselines	281

# Listings

---

## ACRONYMS

---

CNN	Convolutional Neural Network
CNNS	Convolutional Neural Networks
CV	Computer Vision
DP	Dependency Parsing
DNNS	Deep Neural Networks
DRL	Deep Reinforcement Learning
GPUS	Graphics Processing Units
KD	Knowledge Distillation
LSTM	Long Short-Term Memory
LF	Layer Fusion
LM	Language Model
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLM	Masked Language Modeling
MSE	Mean Squared Error
NER	Named Entity Recognition
NLP	Natural Language Processing
NLTG	Natural Language Text Generation
NSP	Neural Sequence Prediction
NSPS	Neural Sequence Predictors
NNRS	Nearest Neighbor Replacement Sampling
NS	Negative Sampling
NMT	Neural Machine Translation
PCA	Principal Component Analysis
RNN	Recurrent Neural Network
RNNS	Recurrent Neural Networks
SGD	Stochastic Gradient Descent

SM	Sequence Modeling
SCNS	Semantically Conditioned Negative Sampling
SOTA	State of The Art
SSL	Self-Supervised Learning
SS	Scheduled Sampling
SVD	Singular Value Decomposition
TPUS	Tensor Processing Units
VRAM	Virtual Random Access Memory

---

## INTRODUCTION

---

Deep Neural Networks (DNNs) are becoming increasingly large, pushing the limits of generalization performance and tackling more complex problems in areas such as Computer Vision (CV), Natural Language Processing (NLP), Robotics and Speech to name a few.

Transformer-based architectures [424] are the current State of The Art (SoTA) in NLP [90, 217, 255, 373, 463] and have begun to supersede Convolutional Neural Networks (CNNs) in CV [329, 445]). These architectures have millions of parameters for each fully-connected layer that make up self-attention blocks [424]. The trend of increased Transformer size is evident from the increase shown on the right plot of Figure 1.1.

For example, a Language Model (LM) such as MegatronLM [388] is made up of a 72-layer GPT-2 model consisting of 8.3 billion parameters, trained with 8-way model parallelism and 64-way data parallelism over 512 Graphics Processing Units (GPUs). Rosset [364] proposed a 17 billion parameter Transformer model for Natural Language Text Generation (NLTG) that consists of 78 layers with a hidden size of 4,256 and each block containing 28 attention heads. The most recent Transformer to date [46] trains a GPT-3 autoregressive language model that contains 175 billion parameters. This model can perform NLP tasks (e.g machine translation, question-answering) and digit arithmetic relatively well with only a few examples, closing the performance gap to similarly large pretrained models that are further fine-tuned for specific tasks and in some cases outperforming them given the large increase in the number of parameters.

Similarly, Convolutional Neural Network (CNN) [115] based architectures [151, 152, 210, 480] used in vision and NLP tasks [121, 170, 200]) have less parameters than Transformers due to the sparse connectivity but are still growing in size. From the left of Figure 1.1, we see that larger overparameterized CNN networks generalize better than smaller networks on large image classification benchmark datasets such as ImageNet. However, recent CNN architectures that aim to reduce the number of Floating Point Operations (FLOPs) and improve training efficiency with less parameters have also shown impressive performance e.g EfficientNet [409].

However, the resources required to train these large CNN and Transformer models on GPUs or Tensor Processing Units (TPUs) let alone use

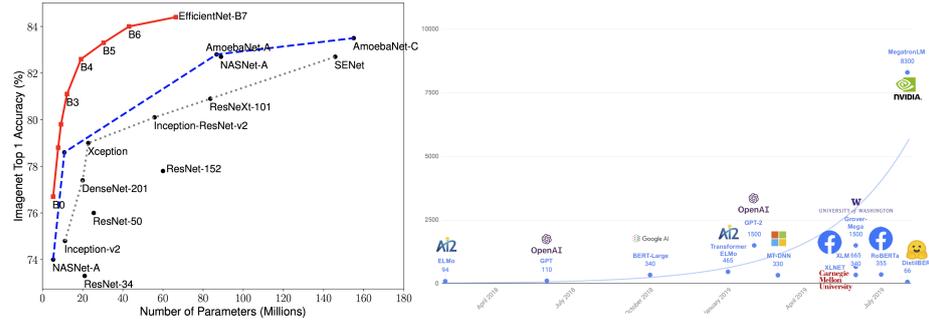


Figure 1.1: Accuracy vs Num. of Parameters for CNN architectures (source on left: Tan and Le [408]) and Num. of Parameters vs Years for Transformers (source on right: Sanh [372])

them for inference is out of reach for a large portion of machine learning practitioners. Moreover, these models have predominantly been driven by improving SoTA models and pushing the boundaries of what complex tasks can be solved using them. Therefore, it is expected that the current trend of increasing network size continues, along with increasing training data size and resource requirements.

**Learning compressed networks can be used to address the aforementioned challenges by reducing parameter count of an already trained model with the aim of maintaining performance close to the original uncompressed network.** These smaller compressed networks can perform better than a model of the same complexity trained from scratch [143] i.e not from a pretrained state but from random initialization.

Hence, **the work presented in this thesis aims to address some limitations of current compression techniques for DNNs and their applications to NLP and CV tasks.** The rest of this introductory chapter is organized as follows. Section 1.1 describes the motivation for this thesis, Section 1.2 outlines the research aims, Section 1.3 describes the research questions, Section 1.4 outlines the research contributions, Section 1.5 lists publications that are peer-reviewed, non-peer reviewed and currently under review and Section 1.6 describes the structure of the thesis and summarizes this introductory chapter.

## 1.1 MOTIVATION

The motivation to compress models has grown and expanded in recent years from being predominantly focused around deployment of small neural networks on mobile devices and FPGAs [144, 153], to the more general aim of learning smaller networks with eased hardware constraints on any device e.g learning on a small number of GPUs and TPUs or the same number of GPUs and TPUs but with a smaller amount of Virtual Random Access Memory (VRAM). Hence, model compression is a critical research endeavour to allow the machine learning community to continue

to deploy, reuse and understand large models under resource constraints. At present, there are clear limitations of various compression methods for creating smaller compressed models, which in turn effects the ease of re-use by other machine learning practitioners. We are motivated to address these limitations with the aim to further reduce memory requirements, retraining and inference time, financial cost and the carbon footprints associated with training larger networks [332]. Below we summarize some of these core challenges in knowledge distillation, pruning and weight sharing approaches that pertain to this thesis.

#### KNOWLEDGE DISTILLATION

1. In **NLP, pretrained input word representations (or an ensemble of representations from different sources) can also be distilled using self-supervised learning**. However, thus far, knowledge distillation has only been carried out on fine-tuned classifiers that use these word representations as input.
2. **Policy-gradient based text generators do not currently explicitly optimize for model-based semantic rewards**, which do not suffer from problems such as sparse rewards and generation length biases of word-overlap metrics (e.g BLEU [324], ROUGE [239]).
3. **Distilling knowledge from a teacher network into a pairwise student model is limited due to poor sample efficiency of negative sampling**. This is because uniformly sampling negatives at random leads to many samples which are too far away from samples of the target class, leading to low informativeness about the class boundary and thus slowing down convergence. By providing methods to improve negative sample efficiency, we improve convergence of contrastive learned models in the knowledge distillation setting.

#### WEIGHT SHARING

1. Pruning sets weights or whole blocks of weights to zero, whereas weight sharing ties weights to the same value prior to training. Tying similar weights instead of pruning is more efficient as the network does not become sparse, which would require specialized sparse hardware to take full advantages of sparse networks. However, it is not clear how weight sharing should be carried out for an already trained model with untied weights. One approach is to dynamically fuse weights that have similar values during retraining in order to preserve layer density and avoid sparsifying layers (as in unstructured pruning), which requires sparse matrix libraries [296] to realize the reduction in parameter count.

## PRUNING

1. The zero-shot generalization of pruned models has not yet been investigated. This is an important research path for understanding how pruned models can be used for more than one task or domain and how particular pruning criteria influence pruned model performance in the zero-shot case.
2. [Chapter 10](#) - Gradient pruning with replacement during fine-tuning of Transformer models has not been explored for stabilizing training and improving convergence from a pretrained state. However, there has been some evidence that successively freezing the gradients of whole layers can improve generalization and training time [168]. We are motivated to investigate online pruning for stabilizing Transformers as they have known stability issues during fine-tuning [250].
3. Pruning techniques do not use outputs from the fine-tuned model as targets during training, only the task provided targets. Pruning could be improved by simultaneously performing knowledge distillation to improve convergence directly after pruning steps, maximize class separability and increase the fidelity between the student and teacher logits.

We are motivated to address the above challenges to improve the compression-generalization tradeoff for various model architectures (e.g. [CNNs](#), Transformers, Recurrent Neural Networks ([RNNs](#))) on binary (e.g. sentiment analysis), multi-class (e.g. language modeling, named entity recognition, part of speech tagging and image classification), pairwise classification (e.g. semantic textual similarity and textual entailment), conditional sequence prediction (e.g. neural machine translation and image captioning) and regression tasks (e.g. word similarity and word analogies) across [NLP](#) and [CV](#) domains.

## 1.2 RESEARCH AIMS

Below we summarize specific research aims ordered by each respective chapter.

1. [Chapter 3](#) - **Sequence predictors suffer from compounding errors** whereby previous predictions are fed back in at the next time step and if the predictions are incorrect it can lead to a cascade of errors. This test time behavior is not reflected in current training schemes and thus we are motivated to propose training schemes which better reflect test time behavior. Mitigating compounding errors has implications for autoregressive models that use previous outputs as inputs and can lead to improvements in training efficiency. The aim is to address this problem by proposing a Nearest Neighbor Replacement Sampling (NNRS)

algorithm to bound compounding errors such that if errors are made they are at least semantically close to the ground truth.

2. **Chapter 4 - Model Compression via Error-Correcting Output Codes.** Computing softmax normalization for high-dimensional outputs, such as language models, can significantly slow down training. In this chapter, we aim to reduce this computation bottleneck by approximating exact softmax normalization using a factored distribution via binary error-correcting output codes.
3. **Chapter 5 - Ensemble-based word embedding compression has only been studied for unsupervised learning or self-supervised learning, but not for supervised learning.** In this chapter, we propose techniques that use task labels to guide how meta-embeddings are reconstructed specifically for the downstream task. This ensures that the meta-embedding input post-training generalizes well as input representations for a particular task or a subset of tasks within a chosen domain. By learning compressed inputs of static<sup>1</sup> word embeddings, we maintain the benefits of ensemble learning but with lower dimensionality comparable to the dimensions of a single word embedding within the ensemble set.
4. **Chapter 6- Current policy-gradient based text generators suffer from sparse reward signals as evaluation metrics such as BLEU and ROUGE only account for word overlaps but not semantic similarity.** This leads to discontinuous rewards (e.g BLEU or ROUGE) when training text generators via reinforcement learning. By addressing this issue using a teacher model that is trained on semantic textual similarity, we aim to optimize and evaluate the similarity between predicted and ground truth passages by backpropogating similarity scores as reward signals in the student text generator.
5. **Chapter 7 - Uniform negative sampling leads to long re-training time in metric learning based KD because many negative samples are non-informative about the class boundaries.** We aim to address this limitation through the use of semantically-conditioned negative sampling to reduce training time with less negative samples than uniformly sampled negatives.
6. **Chapter 8 - Standard structured pruning suffers from high information loss during each pruning step when compared to unstructured pruning,** although they have the benefit of practical computation reduction, unlike unstructured pruning. We aim to address this limitation by *merging* similar weights across layers, as oppose to zeroing out weights. Hence, the information loss will be reduced as the total number of shared weights have a

---

<sup>1</sup> Static means that the word representations do not change as the context changes.

smaller sum of magnitude changes post weight sharing compared to the sum total of weight magnitude changes after pruning.

7. **Chapter 9 - Compression methods have not yet been evaluated in the zero-shot setting where a model is tested on samples from a distributions which it is not been trained.** This is common for cross-lingual models that are fine-tuned on a downstream task in a single language (e.g English) but then need to generalize given text from different languages. In this chapter, we discuss our proposed regularizers that bias fine-tuning to maintain language alignment that has been inherently learned through masked language modeling during cross-lingual pretraining to mitigate the effects of alignment distortion during compression retraining.
8. **Chapter 10 - Standard LM fine-tuning is known to be unstable particularly when the downstream task has few samples available to learn from** or when the LM is required to generalize to various distributions other than the samples used for fine-tuning e.g zero-shot cross-lingual transfer for cross-lingual models. We aim to address this instability by using gradient sparsification methods that regularize Transformers to prevent them from overfitting and losing the benefits of using self-supervised learning for pretraining.
9. **Chapter 11 - Compression methods such as pruning and KD are usually studied and applied in isolation. We aim to propose methods that combine pruning and KD in a way that complement each other** and push SoTA compression without little additional computation overhead.

## 1.3 RESEARCH QUESTIONS

In this section we emphasize the research questions that this thesis aims to address in chapter order. We begin with the first research question discussed in Chapter [Chapter 3](#).

1. In [Chapter 3](#) we aim to answer whether compounding errors can be reduced using sampling methods and ask the following research question:

*can a neural sequence predictor perform better at test time if we close the gap between training time and test time behavior through the use of neighborhood sampling to semantically bound the errors and mitigate the effects of exposure bias ?*

2. In [Chapter 4](#), we aim to find out whether the softmax can be approximated using error-correcting codes and what is the performance-dimensionality trade-off. Formally, we ask the following question:

*can language model training be made more efficient by approximating the softmax with error-correcting output codes, while maintaining close to performance when using the full softmax ?*

3. In [Chapter 5](#), we aim to identify whether an input of ensemble of pretrained word embeddings can be efficiently compressed for downstream tasks and ask the following research question:

*can we distil an ensemble of static word embeddings such that their single distilled representation is learned through supervised learning, unlike prior meta-embeddings that use self-supervised learning prior to fine-tuning ?*

4. Lightweight pretrained sentence similarity teacher networks can be used to optimize policy-gradient based text generators, as a type of reinforcement-learning based knowledge distillation. In [Chapter 6](#), the following research question is asked.

*can we improve text generation by instead optimizing for sentence similarity between the predicted sequence and the ground truth sequence and does this improve text-generation based knowledge distillation ?*

5. [Chapter 7](#) focuses on identifying whether the negative sampling can be made more efficient for contrastive learners as uniform sampling can be inefficient w.r.t to convergence time. The research question this chapter addresses is the following:

*can pairwise-based knowledge distillation be improved w.r.t convergence time and generalization by biasing negative sampling towards harder negative samples ?*

6. **Chapter 8** focuses on performing neural network compression while maintaining the density of the compressed network. The research question asked is the following:

*For structured compression, can dynamically tying layers result in better performance than other methods such as tensor decomposition, knowledge distillation and structured pruning ?*

7. In **Chapter 9** we question whether iterative pruning leads to significant zero-shot performance degradation and what measures and techniques can be taken to mitigate the drop in performance. The research question is the following:

*does the choice of pruning criteria change in the zero-shot setting in comparison to standard supervised learning and can we improve pruning in the zero-shot setting by using alignment regularizers?*

8. In **Chapter 10**, we aim to add gradient noise by using gradient sparsification when fine-tuning pretrained masked language models. This is to address the following research question:

*Can we improve the performance of pretrained Transformers by stabilizing fine-tuning using gradient sparsification methods ?*

9. In **Chapter 11** we identify whether using self-distillation can improve the performance of iteratively pruned Transformer models and maximize cross-correlation between pruned and unpruned representations. Specifically, this is to address the following research question:

*Can pruning be improved by using self-distillation such that the distillation labels smoothen the loss surface and in turn helps the model recover faster from performance degradation due to pruning ?*

#### 1.4 CONTRIBUTIONS

The primary goal of this thesis is to investigate various techniques for improving the compression and training efficiency of neural networks. To this end, the thesis makes a number of noteworthy contributions as listed below:

1. **A method for mitigating exposure bias that replaces ground truth words and predicted words with their synonyms to ensure that even when the model incorrectly predicts tokens, they are more likely to be bounded within the semantic neighborhood of the ground truth tokens.** Exposure bias is a known problem in sequence predictors whereby

the error accumulates along the chain of predictions as one output is fed back into the model at the next timestep. However, for maximum likelihood training, the model is given the ground truth as input and not the model prediction at the previous timestep, leading to a major discrepancy between how the model is trained and used at test time. This contribution addresses this issue by interpolating between ground truth, predictions and semantic neighbors (as defined by pretrained word embedding similarity) at training time. This work is discussed in [Chapter 3](#).

2. **A novel and flexible method to approximate the softmax using error-correcting codes to reduce parameter count in the output of neural sequence predictors that have high output dimensionality. An extension of this softmax approximation is made for conditional text generation to mitigate the problem of accumulating errors along successive predictions.** The softmax function can be expensive to compute for very high dimensional outputs, hence there has been a lot of interest in the past few decades that aim to reduce the dimensionality of the output layer or avoid learning that requires nonlinear and smooth normalization functions such as the softmax. However, other approaches can be viewed as reducing the dimensionality so much that the generalization performance is degraded too much. This contribution provides a flexible tradeoff between dimensionality and performance via error-correcting code where each bit in the code makes up a codeword that represents a word in the vocabulary. Concretely, outputs are represented as binary codes instead of one-hot encodings and additional bits can be appended as error-checks to allow for a large output capacity. This work is discussed in greater detail in [Chapter 4](#).
3. **A method that reconstructs an ensemble of pretrained word embeddings while simultaneously using the encoder output as input to a downstream task. This is referred to as supervised meta-embeddings and improves over self-supervised learned meta-embeddings and single pretrained embeddings.** This contribution involves a multi-task learning approach to learning word meta-embeddings, treating meta-embedding learning as an auxiliary reconstruction task to improve predictions on a main intrinsic (e.g word similarity, analogy) or extrinsic (e.g PoS, NER) task whereby the meta-embedding layer is a shared representation between tasks. A target auto-encoder is also proposed, which aims to predict another pretrained embedding from the remaining embedding in the ensemble set, having the benefit of being able to predict missing embeddings when the vocabulary sets do not overlap across the different pretrained word embeddings. This addresses a gap in the meta-embedding

literature for task-specific (supervised) meta-embedding learning. This work was published at ECAI, 2018 [307] and a subsequent survey on meta-embeddings was published at IJCAI-ECAI 2022 [42]. This is discussed in [Chapter 5](#).

4. In [Chapter 6](#) - **A knowledge distilled reinforcement learning-based text generator that learns from the semantic similarity score given by a pairwise sentence similarity teacher network that is given the predicted sentence and the ground truth sentence representations as input.** In summary, this contribution addresses some of the pitfalls associated with current text generators that are optimized with word overlap metrics. Unlike, word overlap measures, the proposed training method can deal with predictions and ground truth of varying length and most importantly, can deal with semantic similarity. Hence, even if no word in the prediction matches the ground truth it can still obtain a high semantic similarity, enable greater diversity in prediction and avoiding mode collapse where the model simply memorizes templated prediction patterns.
5. **An improved approach to sampling non-uniform negative sampling for contrastive learning in the supervised setting and knowledge distillation setting.** Although contrastive learning has played an important role in pushing the state of the art in computer vision models recently, uniform negative sampling is redundant and inefficient. This contribution addresses this issue by sampling neighbouring images that are close yet not within the same class using pretrained encoder networks that measure the semantic similarity between images. Secondly, latent mixup is introduced as a simple strategy to form *hard* negative samples. Both proposals consistently outperform previous knowledge distillation methods and improve contrastive learned models in the standard supervised learning setup. This work is discussed in [Chapter 7](#).
6. **A novel method that fuses layers in deep neural networks to achieve structured compression and maintains performs close the original uncompressed network.** Merging the most similar layers during retraining of deep networks leads to competitive performance when compared against the original network, while maintaining layer density. Layer fusion is also competitive with pruning, layer decomposition and knowledge distillation without the use of any additional parameters. Mixing weight matrices during layer fusion performs comparably to layer averaging. We also compared how much compression can be achieved with and without retraining for both tasks and the importance of the number of epochs and compression steps. This

work has been accepted at ACML, 2021 [318] and is at length in [Chapter 8](#).

7. **The first analysis of pruning cross-lingual models, how this effects zero-shot cross-lingual transfer and performance differences to pruning in the SSL setup.** Pruning has been extensively studied but had yet to be studied in the context of zero-shot learning. This contribution addresses this gap in the literature and proposes a weight regularizer that mitigates alignment distortion by minimizing the layer-wise Frobenius norm or unit similarity between the pruned model and unpruned model, avoiding overfitting to single language task fine-tuning. A post-analysis of weight distributions after pruning is also carried out and an analysis of how pruned and unpruned weight distributions differ across module types in Transformers. This work has been accepted as a long paper to the Association of Computational Linguistics (ACL) [310] and is discussed in [Chapter 9](#).
8. **A gradient sparsification method that improves stabilization of Transformers by adding gradient noise during fine-tuning.** The contribution here is GradDrop and its multiple variants, a class of techniques that add noise during gradient descent by turning off some gradients within the network in an attempt to stabilize the network. Specifically, epochwise- and layerwise- gradient dropout consistently outperforms standard fine-tuning, gradual unfreezing and other gradient dropout variants. Gradient dropout also significantly improves fine-tuning performance for under-resourced languages. This work is discussed in [Chapter 10](#).
9. **A novel pruning method that combines pruning and knowledge distillation to improve pruned model performance.** - Iteratively pruned models suffer from not being able to recover from performance drops directly after pruning steps when the compression of the model becomes increasingly high. One reason for this is that the loss landscape becomes harder for gradient descent to navigate and find flat local minima given the large reduction of fine-tunable parameters. This contribution involves simultaneously using self-distillation to help highly pruned models recover faster from pruning steps and also improve the class separability. This confluence of pruning and self-distillation is extensively studied for masked language models and its utility on downstream tasks in both monolingual and multi-lingual settings. This work has been accepted to ECML-PKDD, 2022 [309] and discussed in [Chapter 11](#).

## 1.5 PUBLICATIONS AND CODE

The majority of the material in this thesis has been published in [NLP](#) and Machine Learning ([ML](#)) peer-reviewed conferences, while the remaining material is hosted on [arxiv.org](#). Below lists each of these publications in order of year.

- [James O’ Neill](#), Sourav Dutta and Haytham Assem: *AlignReg: Weight Regularizers for Improved Zero-Shot Pruning*, 60<sup>th</sup> Association of Computational Linguistics (ACL), June, 2022 [Chapter 9](#)
- [James O’ Neill](#), Sourav Dutta and Haytham Assem: *Gradient Sparsification For Improved Fine-Tuning of Pretrained Transformers*, arXiv preprint arXiv:2102.06603, July, 2022. *under review* [Chapter 10](#)
- [James O’ Neill](#), Sourav Dutta and Haytham Assem: *Self-Distilled Pruning For Deep Neural Networks*, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD), September, 2022. [Chapter 11](#)
- Danushka Bollegala, [James O’ Neill](#): *A Survey on Word Meta-Embedding Learning*, The 31<sup>st</sup> International Joint Conference on Artificial Intelligence, July, 2022.
- [James O’ Neill](#), Polina Rozenshtein, Ryuichi Kiryo, Motoko Kubota and Danushka Bollegala: *I Wish I Would Have Loved This One, But I Didn’t—A Multilingual Dataset for Counterfactual Detection in Product Reviews*, Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP), November, 2021.[code](#).
- [James O’ Neill](#), Greg Ver Steeg and Aram Galstyan: *Layer-Wise Neural Network Compression via Layer Fusion*, 13<sup>th</sup> Asian Conference on Machine Learning (ACML), November 2021. [Chapter 8](#)
- [James O’ Neill](#) and Danushka Bollegala: *Semantically-Conditioned Negative Samples for Efficient Contrastive Learning*, arXiv preprint arXiv:2102.06603, February, 2021.*under review* [Chapter 7](#)
- Angrosh Mandya, [James O’ Neill](#), Danushka Bollegala and Frans Coenen: *Do not let the history haunt you—Mitigating Compounding Errors in Conversational Question Answering*, Proceedings of the 12<sup>th</sup> Language Resources and Evaluation Conference (LREC), July, 2020. [Chapter 3](#)
- [James O’ Neill](#) and Danushka Bollegala: *Meta-embedding as auxiliary task regularization*, Proceedings of the 24<sup>th</sup> European Conference on Artificial Intelligence (ECAI), August, 2020. [Chapter 5](#)

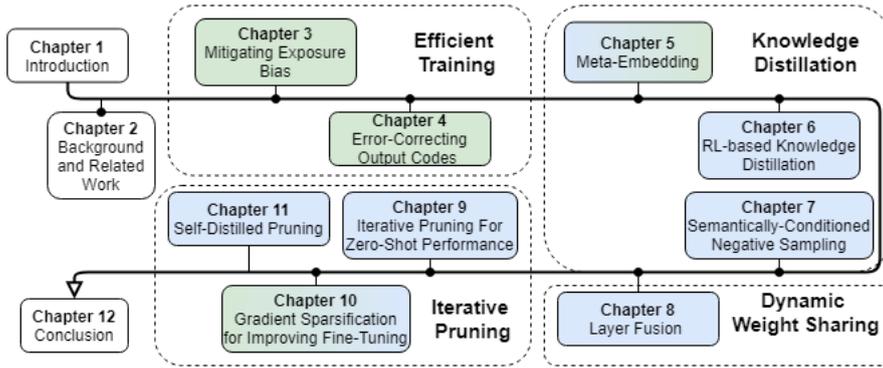


Figure 1.2: Thesis Diagram. Green indicates chapters focused on efficient training, blue is for chapters on model compression and gradients of both indicates chapters which pertain to both efficient training and model compression.

- [James O’ Neill](#) and Danushka Bollegala: *k-Neighbor Based Curriculum Sampling for Sequence Prediction*, arXiv preprint arXiv:2101.09313, November, 2018. [Chapter 3](#). [code](#).
- [James O’ Neill](#) and Danushka Bollegala: *Learning to Evaluate Neural Language Models*, Proceedings of the 16<sup>th</sup> International Conference of the Pacific Association for Computational Linguistics (PACLING), October, 2019.
- [James O’ Neill](#) and Danushka Bollegala: *Error-Correcting Neural Sequence Prediction*, arXiv preprint arXiv:1901.07002, April, 2019. [Chapter 4](#)
- [James O’ Neill](#) and Danushka Bollegala: *Transfer Reward Learning for Policy Gradient-Based Text Generation*, arXiv preprint arXiv:1909.03622, April, 2019. [Chapter 6](#)

## 1.6 THESIS OUTLINE AND SUMMARY

This introductory chapter has given an overview of the research undertaken in this thesis, a description of research aims and motivations, questions, contributions and an outline of the structure of proceeding chapters.

[Figure 1.2](#) summarizes the order of each chapter and how groups of chapters represent efficient training or different model compression topics. Chapters are chronologically ordered which is also conveniently grouped by work on training efficiency ([Chapter 3](#) and [Chapter 4](#)) and proposed model compression techniques such as knowledge distillation ([Chapter 5](#), [Chapter 6](#) and [Chapter 7](#)), weight sharing ([Chapter 8](#)) and iterative pruning ([Chapter 9](#), [Chapter 10](#) and [Chapter 11](#)). In this thesis, we define efficient training methods as methods that reduce

training time and computation when training models from random initialization. [Chapter 5](#) and [Chapter 10](#) cover methods that both improve training efficiency *and* reduce the parameter count as in model compression, hence, they are coloured both green and blue. [Chapter 5](#) to [Chapter 8](#) discusses proposed methods in knowledge distillation and dynamic weight sharing, both of which lead to smaller models that are non-sparse. In the final section of the thesis, from [Chapter 9](#) to [Chapter 11](#), we discuss unstructured pruning methods that lead to sparsely compressed models.

We now continue to [Chapter 2](#) to provide an overview of the vast literature on neural network compression methods consisting of weight sharing, pruning, tensor decomposition, knowledge distillation and quantization. This will set the context and provide the necessary background for understanding the remainder of the thesis. Note that [Chapter 2](#) provides a *general* overview on the compression literature. In subsequent chapters, we will also provide literature reviews aimed specifically for the work discussed in those chapters.

# 2

---

## BACKGROUND AND RELATED WORK

---

In this chapter, we provide an overview of the literature for weight sharing (Section 2.1), pruning (Section 2.2), tensor decomposition (Section 2.3), knowledge distillation (Section 2.4) and quantization (Section 2.5). We begin by concisely defining these compression techniques and provide general remarks on compression, before delving into the related work in each respective area.

1. **Weight Sharing:** Ties weights of different units or layers and averages their gradients during gradient descent.
2. **Pruning:** Zeros out weights in DNNs, either sparsely (unstructured pruning) or whole structured modules (structured pruning).
3. **Matrix and Tensor Decomposition:** Reduces the dimensionality of layers by applying dimensionality reduction techniques (e.g Principal Component Analysis (PCA), Singular Value Decomposition (SVD)) on tensors representing parameters.
4. **Knowledge Distillation:** Trains a smaller student network using the logits or intermediate representations of a larger already trained teacher network.
5. **Quantization:** Reduces the number of bits used to represent weights, activation functions and/or gradient updates. This can also be applied during retraining, which is known as *iterative quantization*.

Retraining is often required to account for some performance loss when applying any of the above compression techniques. The retraining step can be carried out using unsupervised (including self-supervised) learning (e.g tensor decomposition) and supervised learning (e.g knowledge distillation). Unsupervised compression is often used to create compressed models that generalize to a large range of tasks. Alternatively, supervision can be used to gear the compressed model towards a subset of tasks in which case the target task labels are used as opposed to the original data the model was trained on, unlike unsupervised model compression. In some cases, reinforcement learning has also been used to maintain compressed model performance as an alternative to standard supervised learning for iterative pruning [241], knowledge distillation [17] and quantization [464].

With this synopsis of compression we now review prior work, starting with weight sharing.

## 2.1 WEIGHT SHARING

The simplest form of network reduction involves sharing weights between layers or structures within layers (e.g filters in CNNs). Unlike compression techniques discussed in later sections (Section 3-6), standard weight sharing is carried out prior to training the original networks as opposed to compressing the model after training. However, recent work that is discussed here [20, 60, 423] have also been used to reduce DNNs post-training and hence this section is devoted to this straightforward and commonly used technique.

Weight sharing reduces the network size and avoids sparsity. It is not always clear how many and what group of weights should be shared before there is an unacceptable performance degradation for a given network architecture and task. For example, Inan, Khosravi, and Socher [179] find that tying the input and output representations of words leads to good performance while dramatically reducing the number of parameters proportional to the size of the vocabulary of given text corpus. Although, this may be specific to language modelling, since the output classes are a direct function of the inputs which are typically very high dimensional (e.g typically greater than  $10^6$ ). Moreover, this approach requires the embedding matrix to be shared, as opposed to sharing individual or sub-blocks of the matrix. Other approaches include clustering weights such that their centroid is shared among each cluster and using weight penalty term in the objective to group weights in a way that makes them more amenable to weight sharing. These approaches are discussed below along with other recent techniques that have shown promising results when used in DNNs.

### 2.1.1 Clustering-based Weight Sharing

Choosing which weights to share can be achieved through the use of clustering algorithms, where the weights that are within the same cluster are tied. Nowlan and Hinton [316] proposed a soft weight sharing scheme by learning a Gaussian Mixture Model that assigns groups of weights to a shared value given by the mixture model. By using a mixture of Gaussians, weights with high magnitudes that are centered around a broad Gaussian component are penalized less whereas a Gaussian assigned to a subset of parameters with lower variance will assign higher probability density to each parameter in that subset. Hence, the amount of tied parameters varies depending on their clusterability and weight magnitudes.

Equation 2.1 shows the cost function for the Gaussian mixture model where  $p(w_j)$  is the probability density of a Gaussian component with

mean  $\mu_j$  and standard deviation  $\sigma_j$ . Gradient Descent is used to optimize  $w_i$  and mixture parameters  $\pi_j, \mu_j, \sigma_j$  and  $\sigma_y$  where  $K$  is a factor that adjusts the effective number of degrees of freedom,  $y_c$  is a sample within the  $c$ -th cluster and  $d_c$  is the corresponding cluster centroid sample.

$$C = \frac{K}{\sigma_y^2} \sum_c (y_c - d_c)^2 - \sum_i \log \left[ \sum_j \pi_j p_j(w_i) \right] \quad (2.1)$$

The expectation maximization (EM) algorithm is used to optimize these mixture parameters. The number of parameters tied is then proportional to the number of mixture components that are used in the Gaussian model.

AN EXTENSION OF SOFT-WEIGHT SHARING Ullrich, Meeds, and Welling [423] build on soft-weight sharing [316] using a mixture model with factorized posteriors by optimizing the objective in Equation 2.2. Here,  $\tau = 5e^{-3}$  controls the influence of the log-prior means  $\mu$ , variances  $\sigma$  and mixture coefficients  $\pi$ , which are learned during retraining apart from the  $j$ -th component that are set to  $\mu_j = 0$  and  $\pi_j = 0.99$ . Each mixture parameter has a learning rate of  $5 \times 10^{-4}$ . Given the sensitivity of the mixtures to collapsing if the correct hyperparameters are not chosen, they also consider the inverse-gamma hyperprior for the mixture variances, which was shown to stabilize training.

$$\begin{aligned} \mathcal{L}(w, \{\mu_j, \sigma_j, \pi_j\}_{j=0}^J) &= \mathcal{L}_E + \tau \mathcal{L}_C = \\ & - \log p(\tau | X, w) - \tau \log p(w, \{\mu_j, \sigma_j, \pi_j\}_{j=0}^J) \end{aligned} \quad (2.2)$$

After training with the above objective, if the components have a KL-divergence under a set threshold, some of these components are merged [1] given by Equation 2.3. Each weight mapped to the mean of the component with the highest mixture value  $\arg \max(\pi)$ , further performing GMM-based quantization.

$$\pi_{\text{new}} = \pi_i + \pi_j, \quad \mu_{\text{new}} = \frac{\pi_i \mu_i + \pi_j \mu_j}{\pi_i + \pi_j}, \quad \sigma_{\text{new}}^2 = \frac{\pi_i \sigma_i^2 + \pi_j \sigma_j^2}{\pi_i + \pi_j} \quad (2.3)$$

In their experiments, 17 Gaussian components were merge to 6 quantization components, while still leading to performance close to the original LeNet classifier [222] used on MNIST [220]. Although, this method performs quantization, it can also be viewed as weight sharing since the GMM identifies parameters to share the same parameter value.

### 2.1.2 Learning Weight Sharing

This subsection describes work that use regularizers during training that encourage weights to share the same value. Chen et al. [60] use

hash functions to randomly group weight connections into hash buckets that all share the same weight value. Parameter hashing [386, 439] can easily be used with backpropagation whereby each bucket parameters have subsets of weights that are randomly tied i.e each weight matrix contains multiple weights of the same value (referred to as a *virtual matrix*), unlike standard weight sharing where all weights in a matrix are shared between layers.

Zhang et al. [483] explicitly try to learn which weights should be shared by imposing a group order weighted  $\ell_1$  (GrOWL) sparsity regularization term while simultaneously learning to group weights and assign them a shared value. In a given compression step, groups of parameters are identified for weight sharing using the aforementioned sparsity constraint and then the DNN is retrained to fine-tune the structure found via weight sharing. GrOWL first identifies the most important weights and then clusters correlated features to learn the values of the closest important weight throughout training. This can be considered as an adaptive weight sharing technique.

Plummer et al. [343] propose Neural Parameter Allocation Search (NPAS) that learns which parameter groupings to share and can be shared for layers of different size and features of different modality. They find weight sharing with distillation further improves performance for image classification, image-sentence retrieval and phrase grounding.

### 2.1.3 Weight Sharing in Large Architectures

APPLICATIONS IN TRANSFORMERS Dehghani et al. [84] propose Universal Transformers (UT) to combine the benefits of RNNs [165, 367] (recurrent inductive bias) with Transformers [424] (parallelizable self-attention and its global receptive field). Weight sharing is used in UT to reduce the network size and showed strong results on NLP defacto benchmarks while being competitive to the original non-weight shared Transformer.

Dabre and Fujita [77] use a 6-hidden layer Transformer network for neural machine translation (NMT) where the same weights are fed back into the same attention block recurrently. This straightforward approach surprisingly showed similar performance of an untied 6-hidden layer for standard NMT benchmark datasets.

Xiao et al. [454] use shared attention weights in Transformer as dot-product attention can be slow during the auto-regressive decoding stage. Attention weights from hidden states are shared among adjacent layers, drastically reducing the number of parameters proportional to number of attention heads used. The Jensen-Shannon (JS) divergence is taken between self-attention weights of different heads and they average them to compute the average JS score. They find that the weight distribution is similar for layers 2-6 but larger variance is found among encoder-decoder attention although some adjacent layers still

exhibit relatively JS score. Weight matrices are shared based on the JS score whereby layers that have JS score larger than a learned threshold (dynamically updated throughout training) are shared. The criterion used involves finding the largest group of attention blocks that have similarity above the learned threshold to maximize largest number of weight groups that can be shared while maintaining performance. They find a 16 time storage reduction over the original Transformer while maintaining competitive performance.

**DEEP EQUILIBRIUM MODEL** Bai, Kolter, and Koltun [20] propose deep equilibrium models (DEMs) that use a root-finding method to find the equilibrium point of a network and can be analytically backpropogated through at the equilibrium point using implicit differentiation. This is motivated by the observation that hidden states of sequential models converge towards a fixed point. Regardless of the network depth, the approach only requires constant memory because backpropogration only needs to be performed on the layer of the equilibrium point.

On WikiText-103, they show that DEMs can improve SoTA sequence models and reduce memory by 88% use for similar computational requirements as the original models.

#### 2.1.4 *Reusing Layers Recursively*

Recursively re-using layers is another form of weight sharing. This involves feeding the output of a layer back into its input.

Eigen et al. [100] have used recursive layers in CNNs and analyse the effects of varying the number of layers, features maps and parameters independently. They find that increasing the number of layers and number of parameters are the most significant factors while increasing the number of feature maps (i.e the representation dimensionality) improves as a byproduct of the increase in parameters. From this, they conclude that adding layers without increasing the number of parameters can increase performance and that the number of parameters far outweighs the feature map dimensions with respect to performance.

Köpüklü et al. [205] have also focused on reusing convolutional layers using recurrency, while applying batch normalization and channel shuffling after recursed layers to allow filter outputs to be passed as inputs to other filters in the same block. By channel shuffling, the LRU blocks become robust to learning with more than one type of channel, leading to improved performance without increasing the number of parameters. Savarese and Maire [376] learn a linear combination of parameters from an external group of templates. They too use recursive convolutional blocks as apart of the learned parameter shared weighting scheme.

However, layer recursion can lead to *vanishing or exploding gradients* (VEGs). Hence, previous work has aimed to mitigate VEGs in parameter shared networks, namely ones which use the aforementioned recursivity.

Kim, Kwon Lee, and Mu Lee [199] have used residual connections between the input and the output reconstruction layer to avoid signal attenuation, which can further lead to vanishing gradients in the backward pass. This is applied in the context self-supervision by reconstructing high resolution images for image super-resolution. Tai, Yang, and Liu [407] extend the work of Kim, Kwon Lee, and Mu Lee [199]. Instead of passing the intermediate outputs of a shared parameter recursive block to another convolutional layer, they use an elementwise addition of the intermediate outputs of the residual recursive blocks before passing to the final convolutional layer. The original input image is then added to the output of last convolutional layer which corresponds to the final representation of the recursive residual block outputs. Zhang et al. [486] combine residual (skip) connections and dense connections, where skip connections add the input to each intermediate hidden layer input.

Guo et al. [134] address VGs in recursive convolutional blocks by using a gating unit that chooses the number of self-loops for a given block before VEGs occur. They use the Gumbel Softmax trick [183] without gumbel noise to make deterministic predictions of the number of self-loops there should be for a given recursive block throughout training. They also find that batch normalization is at the root of gradient explosion because of the statistical bias induced by having a different number of self-loops during training, effecting the calculation of the moving average. This is addressed by normalizing inputs according to the number of self-loops which is dependent on the gating unit. When used in Resnet-53 architecture, dynamically recursivity outperforms the larger ResNet-101 while reducing the number parameters by 47%.

## 2.2 NETWORK PRUNING

Pruning weights is a commonly used technique to reduce the number of parameters in pretrained DNNs. Pruning can lead to a reduction of storage and model runtime and performance is usually maintaining by retraining the pruned network. Iterative weight pruning prunes while retraining until the desired network size and accuracy tradeoff is met. Pruning is usually not carried out at random, but selected such that *unimportant* weights are removed. In the context of DNNs, random pruning can be detrimental to the models performance and requires more retraining steps to account for the removal of important weights or neurons when compared to well-informed pruning criteria [475].

The simplest pruning criteria involves setting a threshold  $\gamma$  that prunes based on the lowest absolute value (LAV) of weight magnitudes (or the absolute sum of magnitudes of incoming weights for a unit) [137]. Magnitude-based pruning (MBP) can be performed layerwise or globally.

For global MBP, the weights of all layers are first pooled together and vectorized prior to choosing the percentage of LAVs. MBP is the most commonly used in DNNs due to its simplicity and performs well for a wide class of machine learning models on a diverse range of tasks [382]. Alternatively, pruning the weights with LAV of the normalized gradient multiplied by the weight magnitude [227] for a given set of mini-batch inputs can be used, either layer-wise or globally too.

In general, global MBP tends to outperform layer-wise MBP [137, 191, 227, 355] due to its flexibility on the amount of sparsity assigned for each layer, allowing more salient layers to preserve more weight while less salient layers contain more zero weight entries. However, this may not be the case when certain architectural modules are used that lead to discrepancies in weight norms (e.g layer normalization in self-attention blocks) which lead to global MBP over-pruning layers with low weight norm and in turn, tends towards layer collapse for these layers (i.e when two layers of a DNN are completely disconnected).

MBP has also been combined with other strategies such as adding new neurons during iterative pruning to further improve performance [142, 306], where the number of new neurons added is less than the number pruned in the previous pruning step. Hence, the overall number of parameters monotonically decreases over consecutive pruning steps.

Before discussing more involved pruning methods, below lists some important categorical distinctions.

### 2.2.1 *Categorizing Pruning Techniques*

Below, we provide categorization of various pruning techniques.

1. **MBP** - prunes weights with the LAV based on a set threshold or percentage, layer-wise or globally.
2. **Regularization-based Pruning** - penalizes the objective with a regularization term to force the model to learn a network with (e.g  $\ell_1$ ,  $\ell_2$  or lasso weight regularization) smaller weights and prune the smallest weights.
3. **Loss Sensitivity** - compute the sensitivity of the loss function when weights are removed and remove the weights that result in the smallest change in loss.
4. **Search-based approaches** (e.g particle filters, evolutionary algorithms, reinforcement learning) - learn or adapt a set of weights to links or paths within the neural network and keep those which are the most salient for the task. Unlike (1) and (2), the pruning criteria does not rely on Stochastic Gradient Descent (SGD), with the exception of reinforcement learning based pruning.

UNSTRUCTURED VS STRUCTURED PRUNING Another important distinction to be made is that between structured and unstructured pruning. Structured pruning aims to preserve network density for computational efficiency (faster computation at the expense of less flexibility) by removing *blocks* of weights, whereas unstructured is unconstrained to which weights or activations are removed. Hence, sparsity in unstructured pruning techniques provides good generalization performance at the expense of slower computation because sparse tensors have the same dimensionality dense tensors. MBP produces a sparse network that requires sparse matrix multiplication (SMP) libraries to take full advantage of the memory reduction and speed benefits for inference. However, SMP is generally slower than dense matrix multiplication and therefore there has been work towards preserving subnetworks which omit the need for SMP libraries (discussed in [Section 2.2.4](#)).

With these categorical distinctions we now move on to the following subsections that describe various pruning approaches, beginning with regularization-based pruning.

MOTIVATING PRUNING Before the age of [DNNs](#), Castellano, Fanelli, and Pelillo [56] found that training a smaller shallow network from random initialization has poorer generalization compared to an equivalently sized pruned network from a larger pretrained network. More recently, Zhu and Gupta [494] have addressed whether many prior work that reported large reductions in pretrained networks using pruning were already severely overparameterized and could be achieved by simply training a smaller model equivalent in size to the pruned network. They found that deep CNNs and LSTMs, large sparsely pruned networks consistently outperform smaller dense models, achieving a compression ratio of 10 in the number of non-zero parameters with minuscule losses in accuracy. Even when the pruned network is not necessarily overparameterized in the pretraining stage, it still produces consistently better performance than an equivalently sized network trained from scratch [241].

Essentially, having a DNN with larger capacity (i.e more degrees of freedom) allows for a larger set of solutions to be chosen from the parameter space. Overparameterization has also been shown to have a smoothening effect on the loss space [235] when trained with [SGD](#) [73, 99], in turn producing models that generalize better than smaller models. This has been reinforced recently for [DNNs](#) after the discovery of the double descent phenomena [29], whereby a  $2^{nd}$  descent in the test error is found for overparameterized [DNNs](#) that have little to no training errors, occurring after the (critical regime) region where the test error is initially high. This  $2^{nd}$  descent in test error tends to converge to an error lower than that found in the  $1^{st}$  descent where the  $1^{st}$  descent corresponds to the traditional bias-variance tradeoff. Moreover, the norm of the weights becomes dramatically smaller in each layer during this  $2^{nd}$

descent, during the compression phase [390]. Since the weights tend to be close to zero when trained far into this 2<sup>nd</sup> region, it becomes clearer why compression techniques such as pruning have less effect on the networks behaviour when trained to convergence since the magnitude of individual weights becomes smaller as the network grows larger.

Frankle and Carbin [111] also showed that training a network to convergence with more parameters makes it easier to find a subnetwork that maintains performance when trained from scratch, further suggesting that compressing large pretrained overparameterized DNNs that are trained to convergence has advantages from performance and storage perspective over training and equivalently smaller DNN. Even in cases when the initial compression causes a degradation in performance, retraining the compressed model can and is commonly carried out to maintain performance. Below describes the first type of iterative pruning, which relies on weight regularization to encourage weights be close to 0 for MBP.

### 2.2.2 Pruning using Weight Regularization

Regularization-based pruning uses a penalty term that encourages weights to go 0 and during pruning steps. MBP can then be applied. Equation 2.4 shows the commonly used  $\ell_2$  penalty that penalizes large weights  $w_m$  in the  $m$ -th hidden layer with a large magnitude, where  $\mathbf{v}_m \in \mathbb{R}^C$  is the output layer weights.

$$C(\mathbf{w}, \mathbf{v}) = \frac{\epsilon}{2} \left( \sum_{m=1}^h \sum_{l=1}^n \mathbf{w}_{ml}^2 + \sum_{m=1}^h \sum_{p=1}^C \mathbf{v}_{pm}^2 \right) \quad (2.4)$$

However, the main issue with using the above quadratic penalty is that all parameters decay exponentially at the same rate and disproportionately penalizes larger weights. Therefore, Weigend, Rumelhart, and Huberman [438] proposed the objective shown in Equation 2.5. When  $f(w) := w^2/(1+w^2)$  this penalty term is small and when large it tends to 1. Therefore, these terms can be considered as approximating the number of non-zero parameters in the network.

$$C(\mathbf{w}, \mathbf{v}) = \frac{\epsilon}{2} \left( \sum_{m=1}^h \sum_{l=1}^n \frac{\mathbf{w}_{ml}^2}{1 + \mathbf{w}_{ml}^2} + \sum_{m=1}^h \sum_{p=1}^C \frac{\mathbf{v}_{pm}^2}{1 + \mathbf{v}_{pm}^2} \right) \quad (2.5)$$

The derivative  $f'(w) = 2w/(1+w^2)^2$  computed during backpropagation does not penalize large weights as much as Equation 2.4. However, in the context of recent years, where large overparameterized network have shown better generalization when the weights are close to 0. One conjecture is that perhaps Equation 2.5 is more useful in the underparameterized regime. The  $\epsilon$  controls how the small weights decay

faster than large weights. However, the problem of not distinguishing between large weights and very large weights is also an issue. Therefore, Weigend, Rumelhart, and Huberman [438] further propose the objective in Equation 2.6.

$$C(\mathbf{w}, \mathbf{v}) = \epsilon_1 \sum_{m=1}^h \left( \sum_{l=1}^n \frac{\beta \mathbf{w}_{ml}^2}{1 + \beta \mathbf{w}_{ml}^2} + \sum_{p=1}^C \frac{\beta \mathbf{v}_{pm}^2}{1 + \beta \mathbf{v}_{pm}^2} \right) + \epsilon_2 \sum_{m=1}^h \left( \sum_{l=1}^n \mathbf{w}_{ml}^2 + \sum_{p=1}^C \mathbf{v}_{pm}^2 \right) \quad (2.6)$$

Wan et al. [431] have proposed a Gram-Schmidt (GS) based variant of backpropagation whereby GS determines which weights are updated and which ones remain frozen at each epoch.

Li et al. [234] prune filters in CNNs by identifying filters which contribute least to the overall accuracy, defined as the filters that have the lowest sum of weight magnitudes. For a given layer, the sum of the weight magnitudes are computed and since the number of channels is the same across filters, this quantity represents the average of weight value for each kernel. Kernels with weights that have small weight activations will have weak activation and hence these will be pruned. This simple approach leads to less sparse connections and leads to 37% accuracy reduction on average across the models tested while still being close to the original accuracy.

### 2.2.3 Pruning via Loss Sensitivity

Networks can also be pruned by measuring the importance of weights or units by quantifying the change in loss when a weight or unit is removed and pruning the weights that result in the least change in the loss. Many methods from previous decades have been proposed based on this principle [148, 223, 355]. We briefly describe each one below in chronological order.

**SKELETONIZATION** Mozer and Smolensky [299] estimate which units are least important and deletes them during training. The method is referred to as *skeletonization* because it only keeps the units which preserve the main structure of the network that is required for maintaining good generalization performance. Each weight  $w$  in the network is assigned an importance weight  $\alpha$ , where  $\alpha = 0$  corresponds to a redundant hidden unit and  $\alpha = 1$  acts as a standard hidden unit.

To obtain the importance weight for a unit, they calculate the loss derivative with respect to  $\alpha$  as  $\hat{\rho}_i = \partial \mathcal{L} / \alpha_i \Big|_{\alpha_i=1}$  where  $\mathcal{L}$  in this context is the sum of squared errors. Units are then pruned when  $\hat{\rho}_i$  falls below a set threshold. However, they find that  $\hat{\rho}_i$  can fluctuate throughout training and so they propose an exponentially-decayed moving average

over time to smoothen the volatile gradient and also provide better estimates when the squared error is very small. This moving average is given as,

$$\hat{\rho}_i(t+1) = \beta \hat{\rho}_i(t) + (1-\beta) \frac{\partial \mathcal{L}(t)}{\alpha_i} \quad (2.7)$$

where  $\beta = 0.8$  in their experiments. Skeletonization can be applied to current DNNs as it is relatively less costly to other traditional loss-sensitivity based pruning methods, when the moving average window length is small (e.g 2), as  $\alpha$  scales linearly with number of units in the network. However, assigning importance weights for groups of weights, such as filters in a CNN is feasible and aligns with current literature [11, 440] on structured pruning (discussed in Section 2.2.4).

PRUNING WEIGHTS WITH LOW SENSITIVITY Karnin [191] measure the sensitivity  $S$  of the loss function with respect to weights and prune weights with low sensitivity. Instead of removing each weight individually, they approximate  $S$  by the sum of changes experienced by the weight during training as

$$S_{ij} = \left| - \sum_{n=0}^{N-1} \frac{\partial \mathcal{L}}{\partial \mathbf{w}_{ij}} \Delta \mathbf{w}_{ij}(n) \frac{\mathbf{w}_{ij}^f}{\mathbf{w}_{ij}^f - \mathbf{w}_{ij}^i} \right| \quad (2.8)$$

where  $w^f$  is the final weight value at each pruning step,  $w^i$  is the initial weight after the previous pruning step and  $N$  is the number of training epochs. Using backpropagation to compute  $\Delta w$ ,  $\hat{S}_{ij}$  is expressed as,

$$\hat{S}_{ij} = \left| - \sum_{n=0}^{N-1} \left[ \Delta \mathbf{w}_{ij}(n) \right]^2 \frac{\mathbf{w}_{ij}^f}{\nabla(\mathbf{w}_{ij}^f - \mathbf{w}_{ij}^i)} \right| \quad (2.9)$$

If the sum of squared errors is less than that of the previous pruning step and if a weight in a hidden layer with the smallest  $S_{ij}$  changes less than the previous epoch, then these weights are pruned. This is to ensure that weight with small initial sensitivity are not pruned too early, as they may perform well given more retraining steps. If all incoming weights are removed to a unit, the unit is also removed, thus, removing all outgoing weights from that unit. Lastly, they lower bound the number of weights that can be pruned for each hidden layer, therefore, towards the end of training there may be weights with low sensitivity that remain in the network.

VARIANCE ANALYSIS ON SENSITIVITY-BASED PRUNING Engelbrecht [102] remove weights if its variance in sensitivity is not significantly different from zero. If the variance in parameter sensitivities is

not significantly different from zero and the average sensitivity is small, it indicates that the corresponding parameter has little or no effect on the output of the NN over all patterns considered. A hypothesis testing step then uses these variance nullity measures to statistically test if a parameter should be pruned, using the distribution and to test if the expected value of the sensitivity of a parameter over all patterns is equal to zero. The expectation can be written as  $\mathcal{H}_0 : \langle S_{oW,ki} \rangle^2 = 0$  where  $S_{oW}$  is the sensitivity matrix of the output vector with respect to the parameter vector  $\mathbf{W}$  and individual elements  $S_{oW,ki}$  refers to the sensitivity of output to perturbations in parameter over all samples. If the hypothesis is accepted, prune the corresponding weight at the  $(k, i)$  position, otherwise check  $\mathcal{H}_0 : \text{var}(S_{oW,ki}) = 0$  and if this accepted also opt to prune it. They test sum-norm, Euclidean-norm and maximum-norm to compute the output sensitivity matrix. They find that this scheme finds smaller networks than OBD, OBS and standard MBP while maintaining the same accuracy on multi-class classification tasks.

Lauret, Fock, and Mara [218] use a Fourier decomposition of the variance of the model predictions and rank hidden units according to how much that unit accounts for the variance and eliminates based on this variance-based spectral criterion. For a range of variation  $[a_h, b_h]$  of parameter  $w_h$  of layer  $h$  and  $N$  number of training iterations, each weight is varied as  $w_h^{(n)} = (b_h + a_h/2) + (b_h - a_h/2) \sin(\omega_h s^{(n)})$  where  $s^{(n)} = 2\pi n/N$  and  $\omega_h$  is the frequency of  $w_h$  and  $n$  is the training iteration. The  $s_h$  is then obtained by computing the Fourier amplitudes of the fundamental frequency  $\omega_h$ , the first harmonic up to the third harmonic.

### 2.2.3.1 Pruning using Second Order Derivatives

**OPTIMAL BRAIN DAMAGE** As mentioned, deleting single weights is computationally inefficient and slow. LeCun, Denker, and Solla [223] instead estimate weight importance by making a local approximation of the difference in loss between the unpruned and pruned predictions with a Taylor series and use the  $2^{nd}$  derivative of the loss w.r.t the weight. The objective is expressed as Equation 2.10,

$$\delta\mathcal{L} = \sum_i g_i \delta\check{w}_i + \frac{1}{2} \sum_i h_{ii} \delta\check{w}_i^2 + \frac{1}{2} \sum_{i \neq j} h_{ij} \delta\check{w}_i \delta\check{w}_j + \mathcal{O}(\|\check{W}\|^2) \quad (2.10)$$

where  $\check{w}$  are perturbed weights of  $w$ , the  $\delta\check{w}_i$ 's are the components of  $\delta\check{W}$ ,  $g_i$  are the components of the gradient  $\partial\mathcal{L}/\partial\check{w}_i$  and  $h_{ij}$  are the elements of the Hessian  $\mathbf{H}$  where  $\mathbf{H}_{ij} := \partial^2\mathcal{L}/\partial\check{w}_i\partial\check{w}_j$ . Since most well-trained networks will have  $\mathcal{L} \approx 0$ , the  $1^{st}$  term is  $\approx 0$ . Assuming the perturbations on  $W$  are small then the last term will also be small and hence LeCun, Denker, and Solla [223] assume the off-diagonal

values of  $\mathbf{H}$  are 0 and hence  $1/2 \sum_{i \neq j} h_{ij} \delta \check{w}_i \delta w_j := 0$ . Therefore,  $\delta \mathcal{L} \approx \frac{1}{2} \sum_i h_{ii} \delta \check{w}_i^2$ . The  $2^{\text{nd}}$  derivatives  $\mathbf{h}_{kk}$  are calculated by modifying the backpropagation rule. Since  $\mathbf{z}_i = f(\mathbf{a}_i)$  and  $\mathbf{a}_i = \sum_j \mathbf{w}_{ij} \mathbf{z}_j$ , then by substitution  $\frac{\partial^2 \mathcal{L}}{\partial \mathbf{w}_{ij}^2} = \frac{\partial^2 \mathcal{L}}{\partial \mathbf{a}_i^2} z_j$  and they further express the  $2^{\text{nd}}$  derivative of the activation output as,

$$\frac{\partial^2 \mathcal{L}}{\partial \mathbf{a}_i^2} = f'(\mathbf{a}_i)^2 - \sum_l \mathbf{w}_{li}^2 \frac{\partial^2 \mathcal{L}}{\partial \mathbf{a}_l^2} - f''(\mathbf{a}_i)^2 \frac{\partial^2 \mathcal{L}}{\partial \mathbf{z}_i^2} \quad (2.11)$$

The derivative of the mean squared error with respect to the to the last linear layer output is then

$$\frac{\partial^2 \mathcal{L}}{\partial \mathbf{a}_i^2} = 2f'(\mathbf{a}_i)^2 - 2(\mathbf{y}_i - \mathbf{z}_i) f''(\mathbf{a}_i) \quad (2.12)$$

The importance of weight  $w_i$  is then  $s_k \approx h_{kk} \mathbf{w}_k^2 / 2$  and the portion of weights with lowest  $s_k$  are iteratively pruned during retraining.

**OPTIMAL BRAIN SURGEON** Hassibi, Stork, and Wolff [148] improve over OBD by preserving the off diagonal values of the Hessian, showing empirically that these terms are actually important for pruning and assuming a diagonal Hessian hurts pruning accuracy.

To make this Hessian computation feasible, they exploit the recursive relation for calculating the inverse hessian  $\mathbf{H}^{-1}$  from training data and the structural information of the network. Moreover, using  $\mathbf{H}^{-1}$  has advantages over OBD in that it does require further re-training post-pruning. They denote a weight to be eliminated as  $w_q = 0$ ,  $\delta w_q + w_q = 0$  with the objective to minimize the following objective:

$$\min_q \left\{ \min_{\delta \mathbf{w}} \left\{ \frac{1}{2} \delta \mathbf{w}^T \cdot \mathbf{H} \cdot \delta \mathbf{w} \right\} \quad s.t. \quad \mathbf{e}_q^T \cdot \mathbf{w} + w_q = 0 \right\} \quad (2.13)$$

where  $\mathbf{e}_q$  is the unit vector in parameter space corresponding to parameter  $w_q$ . To solve Equation 2.14 they form a Lagrangian from Equation 2.13:

$$\mathcal{L} = \frac{1}{2} \delta \mathbf{w}^T \cdot \mathbf{H} \cdot \delta \mathbf{w} + \lambda (\mathbf{e}_q^T \cdot \delta \mathbf{w} + w_q) \quad (2.14)$$

where  $\lambda$  is a Lagrange undetermined multiplier. The functional derivatives are taken and the constraints of Equation 2.13 are applied. Finally, matrix inversion is used to find the optimal weight change and resulting change in error is expressed as,

$$\delta \mathbf{w} = \frac{w_q}{[\mathbf{H}_{qq}^{-1}]} \mathbf{H}^{-1} \mathbf{e}_q \quad \text{and} \quad \mathcal{L}_q = \frac{1}{2} \frac{w_q^2}{[\mathbf{H}_{qq}^{-1}]} \quad (2.15)$$

Defining the first derivative as  $\mathbf{X}_k := \frac{f(\mathbf{x}; \mathbf{W})}{\partial \mathbf{W}}$  the Hessian is expressed as,

$$\mathbf{H} = \frac{1}{P} \sum_{k=1}^P \sum_{j=1}^n \mathbf{X}_{k,j} \mathbf{X}_{k,j}^T \quad (2.16)$$

for an  $n$ -dimensional output and  $P$  samples. This can be viewed as the sample covariance of the gradient and  $\mathbf{H}$  can be recursively computed as,

$$\mathbf{H}_{m+1}^{-1} = \mathbf{H}_m^{-1} + \frac{1}{P} \mathbf{X}_{m+1}^T \cdot \mathbf{X}_{m+1}^T \quad (2.17)$$

where  $\mathbf{H}_0 = \alpha \mathbf{I}$  and  $\mathbf{H}_P = \mathbf{H}$ . Here  $10^{-8} \leq \alpha \leq 10^{-4}$  is necessary to make  $\mathbf{H}^{-1}$  less sensitive to the initial conditions. For OBS,  $\mathbf{H}^{-1}$  is required and to obtain it they use a matrix inversion formula [189] which leads to the following update:

$$\mathbf{H}_{m+1}^{-1} = \mathbf{H}_m^{-1} - \frac{\mathbf{H}_m^{-1} \cdot \mathbf{X}_{m+1} \cdot \mathbf{X}_{m+1}^T \cdot \mathbf{H}_m^{-1}}{P + \mathbf{X}_{m+1}^T \cdot \mathbf{H}_m^{-1} \cdot \mathbf{X}_{m+1}} \quad (2.18)$$

where  $\mathbf{H}_0 = \alpha \mathbf{I}$  and  $\mathbf{H}_P = \mathbf{H}$ . This recursion step is then used as apart of Equation 2.15, can be computed in one pass of the training data  $1 \leq m \leq P$  and computational complexity of  $\mathbf{H}$  remains the same as  $\mathbf{H}^{-1}$  as  $\mathcal{O}(Pn^2)$ . Hassibi, Stork, and Wolff [148] have also extended their work on approximating the inverse hessian [147] to show that this approximation works for any twice differentiable objective (not only constrained to sum of squared errors) using the Fisher's score.

Other methods to Hessian approximation include dividing the network into subsets to use block diagonal approximations and eigen decomposition of  $\mathbf{H}^{-1}$  [148] and principal components of  $\mathbf{H}^{-1}$  [230] (unlike aforementioned approximations, Levin, Leen, and Moody [230] do not require the network to be trained to a local minimum). However the main drawback is that the Hessian is relatively expensive to compute for these methods, including OBD. For  $n$  weights, the Hessian requires  $\mathcal{O}(n^2/2)$  elements to store and performs  $\mathcal{O}(Pn^2)$  calculations per pruning step, where  $P$  is total number of pruning steps.

### 2.2.3.2 Pruning using First Order Derivatives

As  $2^{nd}$  order derivatives are expensive to compute and the aforementioned approximations may be insufficient in representing the full Hessian, other work has focused on using  $1^{st}$  order information as an alternative approximation to inform the pruning criterion.

Molchanov et al. [295] use a Taylor expansion (TE) as a criterion to prune by choosing a subset of weights  $W_s$  which have a minimal change on the cost function. They also add a regularization term that explicitly

regularize the computational complexity of the network. Equation 2.19 shows how the absolute cost difference between the original network cost with weights  $w$  and the pruned network with  $w'$  weights is minimized such that the number of parameters are decreased where  $\|\cdot\|_0$  denotes the 0-norm bounds the number of non-zero parameters  $W_s$ .

$$\min_{\mathbf{W}'} |\mathcal{C}(D|\mathbf{W}') - \mathcal{C}(D|\mathbf{W})| \quad \text{s.t.} \quad |\mathbf{W}'|_0 \leq \mathbf{W}_s \quad (2.19)$$

Unlike OBD, they keep the absolute change  $|y|$  resulting from pruning, as the variance  $\sigma_y^2$  is non-zero and correlated with stability of the  $\partial C / \partial h$  throughout training, where  $\mathbf{h}$  is the activation of the hidden layer. Under the assumption that samples are independent and identically distributed,  $\mathbb{E}(|y|) = \sigma\sqrt{2}/\sqrt{\pi}$  where  $\sigma$  is the standard deviation of  $y$ , known as the expected value of the half-normal distribution. So, while  $y$  tends to zero, the expectation of  $|y|$  is proportional to the variance of  $y$ , a value which is empirically more informative as a pruning criterion.

They rank the order of filters pruned using the TE criterion and compare to an oracle rank (i.e the best ranking for removing pruned filters) and find that it has higher spearman correlation to the oracle when compared against other ranking schemes. This can also be used to choose which filters should be transferred to a target task model. They compute the importance of neurons or filters  $z$  by estimating the mutual information with target variable  $\text{MI}(z; y)$  using information gain  $IG(y|z) = \mathcal{H}(z) + \mathcal{H}(y) - \mathcal{H}(z, y)$  where  $\mathcal{H}(z)$  is the entropy of the variable  $z$ , which is quantized to make this estimation tractable.

**FISHER PRUNING** Theis et al. [413] extend the work of Molchanov et al. [295] by motivating the pruning scheme and providing computational cost estimates for pruning as adjacent layers are successively being pruned. Unlike OBD and OBS, they use, fisher pruning as it is more efficient since the gradient information is already computed during the backward pass. Hence, this pruning technique uses 1<sup>st</sup> order information given by the 2<sup>nd</sup> TE term that approximates the loss with respect to  $w$ . The fisher information is then computed during backpropagation and uses as the pruning criterion.

The gradient can be formulated as Equation 2.20, where  $\mathcal{L}(w) = \mathbb{E}_P[-\log Q_w(y|\mathbf{x})]$ ,  $d$  represents a change in parameters,  $P$  is the underlying distribution,  $Q_w(y|x)$  is the posterior from the model  $H$  is the Hessian matrix.

$$\begin{aligned} g &= \nabla \mathcal{L}(w), \quad \mathbf{H} = \nabla^2 \mathcal{L}(w), \\ \mathcal{L}(w + d) - \mathcal{L}(w) &\approx \mathbf{g}^T d + \frac{1}{2} d^T \mathbf{H} d \quad (2.20) \\ \mathcal{L}(\mathbf{W} - \mathbf{W}_k \mathbf{e}_i) - \mathcal{L}(\mathbf{W}) &+ \beta \cdot (C(W - \mathbf{W}_k \mathbf{e}_i) - C(\mathbf{W})) \end{aligned}$$

**PIGGYBACK PRUNING** Mallya and Lazebnik [270] propose a dynamic masking (i.e pruning) strategy whereby a mask is learned to adapt a dense network to a sparse subnetwork for a specific target task. The backward pass for binary mask is expressed as,

$$\frac{\partial \mathcal{L}}{\partial m_{ji}} = \left( \frac{\partial \mathcal{L}}{\partial \hat{y}_j} \right) \cdot \left( \frac{\partial y_j}{\partial m_{ji}} \right) = \partial \hat{y}_j \cdot w_{ji} \cdot x_i, \quad (2.21)$$

where  $m_{ij}$  is an entry in the mask  $m$ ,  $\mathcal{L}$  is the loss function and  $\hat{y}_j$  is the prediction when the  $j$ -th mask is applied to the weights  $w$ . The matrix  $m$  can then be expressed as  $\frac{\partial \mathcal{L}}{\partial m} = (\delta \mathbf{y} \mathbf{x}^T) \mathbf{W}$ . Note that although the threshold for the mask  $m$  is non-differentiable, but they perform a backward pass anyway. The justification is that the gradients of  $m$  act as a noisy estimate of the gradients of the real-valued mask weights  $m_r$ . For every new task,  $m$  is tuned with a new final linear layer.

#### 2.2.4 Structured Pruning

Since standard pruning leads to sparse weight tensors, structured pruning can be used as alternative to reduce speed and memory by removing whole structures (e.g rows, columns or sub-blocks of a weight tensor). This is favorable for current hardware which is well-suited for dense matrix multiplications and completely removes any zero entries. For architectures that encoded structural inductive biases such as CNNs, structured pruning is well suited to prune away channels and filters. Hence, below describes work that use group-wise regularizers, structured variational, Adversarial Bayesian methods to achieve structured pruning in CNNs.

##### 2.2.4.1 Structured Pruning via Weight Regularization

**GROUP SPARSITY REGULARIZATION** Group sparse regularizers enforce a subset of weight groupings, such as filters in CNNs, to be close to zero when trained using **SGD**. Consider a convolutional kernel represented as a tensor  $K(i, j, s, :)$ , the group-wise  $\ell_2, 1$ -norm is given as

$$\omega_{2,1}(K) = \lambda \sum_{i,j,s} \|\Gamma_{ijs}\| = \lambda \sum_{ijs} \sqrt{\sum_{t=1}^T K(i, j, s, t)^2} \quad (2.22)$$

where  $\Gamma_{ijs}$  is the group of kernel tensor entries  $K(i, j, s, :)$  where  $(i, j)$  are the pixel of  $i$ -th row and  $j$ -th column of the image for the  $s$ -th input feature map and the  $t$ -th channel. This regularization term forces some  $\Gamma_{ijs}$  groups to be close to zero, which can be removed during retraining depending on the amount of compression that the practitioner predefines.

STRUCTURED SPARSITY LEARNING Wen et al. [440] show that their proposed structural regularization can reduce a ResNet architecture with 20 layers to 18 with 1.35 percentage point accuracy increase on CIFAR-10, which is even higher than the larger 32 layer ResNet architecture. They use a group lasso regularization to remove whole filters, across channels, shape and depth.

Equation 2.23 shows the loss to be optimized to remove unimportant filters and channels, where  $\mathbf{W}^{(l)}_{n_l, c_l, :, :}$  is the  $c$ -th channel of the  $l$ -th filter for a collection of all weights  $\mathbf{W}$  and  $\|\cdot\|$  is the group Lasso regularization term where  $\|\mathbf{w}^{(g)}\|_g = \sqrt{\sum_{i=1}^{|\mathbf{w}^{(g)}|} (\mathbf{w}^{(g)})^2}$ ,  $\|\cdot\|_g$  is the group Lasso and  $|\mathbf{w}^{(g)}|$  is the number of weights in  $\mathbf{w}^{(g)}$ . Here  $\lambda_n$  and  $\lambda_c$  are regularization terms both controlling the influence of filter and channel weight regularization respectively.

Since zeroing out the  $l$ -th filter leads to the feature map output being redundant, it results in the  $l + 1$  channel being zeroed as well. Hence, structured sparsity learning is carried out for both filters and channels simultaneously.

$$\begin{aligned} \mathcal{L}(\mathbf{W}) = \mathcal{L}_D(\mathbf{W}) + \lambda_n \cdot \sum_{l=1}^L \left( \sum_{n_l=1}^N \|\mathbf{W}^{(l)}_{n_l, :, :, :}\|_g \right) \\ + \lambda_c \cdot \sum_{l=1}^L \left( \sum_{c_l=1}^{C_l} \|\mathbf{W}^{(l)}_{:, c_l, :, :}\|_g \right) \end{aligned} \quad (2.23)$$

#### 2.2.4.2 Structured Pruning via Loss Sensitivity

STRUCTURED BRAIN DAMAGE The aforementioned OBD has also been extended to remove groups of weights using group-wise sparse regularizers (GWSR) Lebedev and Lempitsky [224]. In the case of filters in CNNs, this results in smaller reshaped matrices, leading to smaller and faster CNNs. The GWSR is added as a regularization term during retraining a pretrained CNN and after a set number of epochs, the groups with the smallest  $\ell_2$  norm are deleted and the number of groups are predefined as  $\tau \in [0, 1]$  (a percentage of the size of the network). However, they find that when choosing a value for  $\tau$ , it is difficult to set the regularization influence term  $\lambda$  and can be time consuming manually tuning it. Moreover when  $\tau$  is small, the regularization strength of  $\lambda$  is found to be too heavy, leading to many weight groups being biased towards 0 but not being very close to it. This results in poor performance as it becomes more unclear what groups should be removed. However, the drop in accuracy due to this can be remedied by further retraining after performing OBD. Hence, retraining occurs on the sparse network without using the GWSR.

### 2.2.4.3 Sparse Bayesian Priors

**SPARSE VARIATIONAL DROPOUT** Seminal work, such as the aforementioned Skeletonization [299] technique has essentially tried to learn weight saliency. Variational dropout (VD), or more specifically Sparse Variational Dropout [(SpVD) 294], learn individual dropout rates for each parameter in the network using variational inference (VI). In Sparse VI, sparse regularization is used to force activations with high dropout rates (unlike the original VD [203] where dropout rates are bound at 0.5) to go to 1 leading to their removal. Much like other sparse Bayes learning algorithms, VD exhibits the Automatic relevance determination (ARD) effect<sup>1</sup>. Molchanov, Ashukha, and Vetrov [294] propose a new approximation to the KL-divergence term in the VD objective and also introduce a way to reduce variance in the gradient estimator which leads to faster convergence. VI is performed by minimizing the bound between the variational Gaussian prior  $q_\phi(w)$  and prior over the weight  $p(w)$  as:

$$\mathcal{L}(\phi) = \max_{\phi} \mathcal{L}_D - D_{\text{KL}}\left(q_\phi(w) \parallel p(w)\right) \quad (2.24)$$

where  $\mathcal{L}_D(\phi) = \sum_{n=1}^N \mathbb{E}_{q_\phi(w)} \left[ \log p(y_n | \mathbf{x}_n, \mathbf{w}_n) \right]$ .

They use the reparameterization trick to reduce variance in the gradient estimator when  $\alpha > 0.5$  by replacing multiplicative noise  $1 + \sqrt{\alpha_{ij}} \cdot \epsilon_{ij}$  with additive noise  $\sigma_{ij} \cdot \epsilon_{ij}$ , where  $\epsilon_{ij} \sim \mathcal{N}(0; 1)$  and  $\sigma_{ij}^2 = \alpha_{ij} \cdot \theta_{ij}^2$  is tuned by optimizing the variational lower bound w.r.t  $\theta$  and  $\sigma$ . This difference with the original VD allow weights with high dropout rates to be removed.

Since the prior and approximate posterior are fully factorized, the full KL-divergence term in the lower bound is decomposed into a sum:

$$D_{\text{KL}}(q(\mathbf{W} | \theta, \alpha) \parallel p(\mathbf{W})) = \sum_{ij} D_{\text{KL}}(q(w_{ij} | \theta_{ij}, \alpha_{ij}) \parallel p(w_{ij})) \quad (2.25)$$

Since the uniform log-prior is an improper prior, the KL divergence is only computed up to an additional constant [203].

In the VD model this term is intractable, as the expectation  $E \sim N(1, \alpha_{ij}) \log |\cdot|_i n$  cannot be computed analytically [203]. Hence, they approximate the negative KL. The negative KL increases as  $\alpha_{ij}$  increases which means the regularization term prefers large values of  $\alpha_{ij}$  and so the correspond weight  $w_{ij}$  is dropped from the model. Since using **SVD** at the start of training tends to drop too many weights early since the weights are randomly initialized, **SVD** is used after an initial pretraining stage and hence this is why it is considered a pruning technique.

<sup>1</sup> Automatic relevance determination provides a data-dependent prior distribution to prune away redundant features in the overparameterized regime i.e more features than samples

BAYESIAN STRUCTURED PRUNING Structured pruning has also been achieved from a Bayesian view [263] of learning dropout rates. Sparsity inducing hierarchical priors are placed over the units of a DNN and those units with high dropout rates are pruned. Pruning by unit is more efficient from a hardware perspective than pruning weights as the latter requires priors for each individual weight, being far more computationally expensive and has the benefit of being more efficient from a hardware perspective as whole groups of weights are removed.

Consider a DNN as  $p(D|w) = \prod_{i=1}^N p(y_i|x_i, w)$  where  $x_i$  is a given input sample with a corresponding target  $y_i$ ,  $w$  are the weights of the network, governed by a prior distribution  $p(w)$ . Since computing the posterior  $p(w|D) = p(D|w)p(w)/p(D)$  explicitly is intractable,  $p(w)$  is approximated with a simpler distribution, such as a Gaussian  $q(w)$ , parameterized by variational parameters  $\phi$ . The variational parameters are then optimized as,

$$\mathcal{L}_E = \mathbb{E}_{q_\phi(w)}[\log p(D|w)], \quad (2.26)$$

$$\mathcal{L}_C = \mathbb{E}_{q_\phi(w)}[\log p(w)] + \mathcal{H}(q_\phi(w)) \quad (2.27)$$

$$L(\phi) = \mathcal{L}_E + \mathcal{L}_C \quad (2.28)$$

where  $\mathcal{H}(\cdot)$  denotes the entropy and  $\mathcal{L}(\phi)$  is known as the evidence-lower-bound (ELBO). They note that  $\mathcal{L}_E$  is intractable for noisy weights and in practice Monte Carlo integration is used. When the simpler  $q_\phi(w)$  is continuous the reparameterization trick is used to backpropagate through the deterministic part  $\phi$  and Gaussian noise  $\epsilon \sim N(0, \sigma^2 I)$ . By substituting this into Equation 2.26 and using the local reparameterization trick [203] they can express  $\mathcal{L}(\phi)$  as

$$\mathcal{L}(\phi) = \mathbb{E}_p(\epsilon)[\log p(D|f(\phi, \epsilon))] + \mathbb{E}_{q_\epsilon(w)}[\log p(w)] + \mathcal{H}(q_\phi(w)) \quad (2.29)$$

where  $w = f(\phi, \epsilon)$ .

with unbiased SGD estimates of the ELBO w.r.t the variational parameters  $\phi$ . They use mixture of a log-uniform prior and a half-Cauchy prior for  $p(w)$  which equates to a horseshoe distribution [55]. By minimizing the negative KL divergence between the normal-Jeffreys scale prior  $p(z)$  and the Gaussian variational posterior  $q_\phi(z)$  they can learn the dropout rate  $\alpha_i = \sigma^2 z_i / \mu_2 z_i$  as

$$-D_{\text{KL}}(\phi(z)||p(z)) \approx A \sum_i (k_1 \sigma(k_2 + k_3 \log \alpha_i) - 0.5m(-\log \alpha_i) - k_1) \quad (2.30)$$

where  $\sigma(\cdot)$  is the sigmoid function,  $m(\cdot)$  is the softplus function and  $k_1 = 0.64$ ,  $k_2 = 1.87$  and  $k_3 = 1.49$ . A unit  $i$  is pruned if

its variational dropout rate does not exceed threshold  $t$ , as  $\log \alpha_i = (\log \sigma^2 z_i - \log \mu_2 z_i) \geq t$ .

This prior parametrization readily allows for a more flexible marginal posterior over the weights as the compound distribution is given as,

$$q_\phi(W) = \int q_\phi(W|z)q_\phi(z)dz \quad (2.31)$$

PRUNING VIA VARIATIONAL INFORMATION BOTTLENECK Dai, Zhu, and Wipf [78] minimize the variational lower bound (VLB) to reduce the redundancy between adjacent layers by penalizing their mutual information to ensure each layer contains useful and distinct information. A subset of neurons are kept while the remaining neurons are forced toward 0 using sparse regularization that occurs as apart of their variational information bottleneck (VIB) framework. They show that the sparsity inducing regularization has advantages over previous sparsity regularization approaches for network pruning.

Equation 2.32 shows the objective for compressing neurons (or filters in CNNs) where  $\gamma_i$  controls the amount of compression for the  $i$ -th layer and  $L$  is a weight on the data term that is used to ensure that for deeper networks the sum of KL factors does not result in the log likelihood term outweighed when finding the globally optimal solution.

$$\mathcal{L} = \sum_{i=1}^L \gamma_i \sum_{j=1}^{r_i} \log \left( 1 + \frac{\mu_{i,j}^2}{\sigma_{i,j}^2} \right) - L \mathbb{E}_{\{\mathbf{x}, \mathbf{y}\} \sim D, \mathbf{h} \sim p(\mathbf{h}|\mathbf{x})} \left[ \log q(\mathbf{y}|\mathbf{h}_L) \right] \quad (2.32)$$

$L$  naturally arises from the VIB formulation unlike probabilistic networks models. The  $\log(1 + u)$  in the KL term is concave and non-decreasing for range  $[0, \infty]$  and therefore favors solutions that are sparse with a subset of parameters exactly zero instead of many shrunken ratios  $\alpha_{i,j} : \mu_{i,j}^2 \sigma_{i,j}^{-2}, \forall i, j$ .

Each layer is sampled  $\epsilon_i \sim \mathcal{N}(0, I)$  in the forward pass and  $\mathbf{h}_i$  is computed. Then the gradients are updated after backpropogation for  $\{\mu_i, \sigma_i \mathbf{W}_i\}_{i=1}^L$  and output weights  $\mathbf{W}_y$ .

The conditional distribution  $p(\mathbf{h}_i|\mathbf{h}_{i-1})$  and  $\mathbf{h}_i$  are sampled by multiplying  $f_i(\mathbf{h}_{i-1})$  with a random variable  $\mathbf{z}_i := \mu_i + \epsilon_i \circ \sigma_i$ . They show that when using VIB network, the mutual information increases between  $\mathbf{x}$  and  $\mathbf{h}_1$  as it initially begins to learn and later in training the mutual information begins to drop as the model enters the compression phase. In contrast, the mutual information for the original stayed consistently high tending towards 1.

GENERATIVE ADVERSARIAL-BASED STRUCTURED PRUNING Lin et al. [243] extend beyond pruning well-defined structures, such as filters, to more general structures which may not be predefined in the

network architecture. They do so applying a soft mask to the output of each structure in a network to be pruned and minimize the mean squared error with a baseline network and also a minimax objective between the outputs of the baseline and pruned network where a discriminator network tries to distinguish between both outputs. During retraining, soft mask weights are learned over each structure (i.e filters, channels, ) with a sparse regularization term (namely, a fast iterative shrinkage-thresholding algorithm) to force a subset of the weights of each structure to go to 0. Those structures which have corresponding soft mask weight lower than a predefined threshold are then removed throughout the adversarial learning. This soft masking scheme is motivated by previous work [242] that instead used hard thresholding using binary masks, which results in harder optimization due to non-smoothness. Although they claim that this sparse masking can be performed with label-free data and transfer to other domains with no supervision, the method is largely dependent on the baseline (i.e teacher network) which implicitly provides labels as it is trained with supervision, and thus its pruned network transferability is largely dependent on this.

### 2.2.5 Search-based Pruning

Search-based techniques can be used to search the combinatorial subset of weights to preserve in DNNs. This section includes pruning techniques that don't rely on gradient-based learning but also evolutionary algorithms and SMC methods.

#### 2.2.5.1 Evolutionary-Based Pruning

**PRUNING USING GENETIC ALGORITHMS** The basic procedure for Genetic Algorithms (GAs) in the context of DNNs is as follows; (1) generate populations of parameters (or *chromosomes* which are binary strings), (2) keep the top-k parameters that perform the best (referred to as tournament selection) according to a predefined *fitness* function (e.g classification accuracy), (3) randomly mix (i.e cross over) between the parameters of different sets within the top-k and perturb a portion of the resulting parameters (i.e mutation) and (4) repeat this procedure until convergence. This procedure can be used to find a subset of the DNN network that performs well.

Whitley, Starkweather, and Bogart [441] use a GA to find the optimal set of weights which involves connecting and reconnecting weights to find mutations that lead to the highest fitness (i.e lowest loss). They define the number of backpropagation steps as  $ND + B$  where  $B$  is the baseline number of steps,  $N$  is the number of weights pruned and  $D$  is the increase in number of backpropagation steps. Hence, if the network is heavily pruned the network is allocated more retraining steps. Unlike standard pruning techniques, weights can be reintroduced if they are apart of combination that leads to a relatively good fitness score. They

assign higher reward to network which more heavily pruned, otherwise referred to as *selective pressure* in the context of genetic algorithms.

Since the cross-over operation is not specific to the task by default, interference can occur among related parameters in the population which makes it difficult to find a near optimal solution, unless the population is very large (i.e exponential with respect to the number of features). Cantu-Paz [52] identify the relationship between variables by computing the joint distribution of individuals left after tournament selection and use this sub-population to generate new members of the population for the next iteration. This is achieved using 3 distribution estimation algorithms (DEA). They find that DEAs can improve GA-based pruning and that in pruned networks using GA-based pruning results in faster inference with little to no difference in performance compared to the original network.

Recently, Hu et al. [172] have pruned channels from a pretrained CNN using GAs and performed knowledge distillation on the pruned network. A kernel is converted to a binary string  $K$  with a length equal to the number of channels for that kernel. Then each channel is encoded as 0 or 1 where channels with a 0 are pruned and the  $n$ -th kernel  $K_n$  is represented a a binary series after sampling each bit from a Bernoulli distribution for all  $C$  channels. Each member (i.e channels) in the population is evaluated and top-k are kept for the next generation (i.e iteration) based on the fitness score where k corresponds to the total amount of pruning. The Roulette Wheel algorithm is used as the selection strategy [125] whereby the  $n$ -th member of the  $m$ -th generation  $I_{m,n}$  has a probability of selection proportional to its fitness relative to all other members. This can simply be implemented by inputting all fitness scores for all members into a softmax. To avoid members with high fitness scores losing information post mutation and cross-over, they also copy the highest fitness scoring members to the next generation along with their mutated versions.

The main contribution is a 2-stage fitness scoring process. First, a local TS approximation of a layer-wise error function using the aforementioned OBS objective [96] (recall that OBS mainly revolves around efficient Hessian approximation) is used sequentially from the first layer to the last, followed by a few epochs of retraining to restore the accuracy of the pruned network. Second, the pruned network is distilled using a cross-entropy loss and regularization term that forces the features maps of the pruned network to be similar to the distilled model, using an attention map to ensure both corresponding layer feature maps are of the same and fixed size. They achieve SoTA on ImageNet and CIFAR-10 for VGG-16 and ResNet CNN architectures using this approach.

PRUNING VIA SIMULATED ANNLEALING Noy et al. [317] propose to reduce search time for searching neural architectures by relaxing the discrete search to continuous that allows for a differentiable simulated

annealing that is optimized using gradient descent (following from the DARTS [249] approach). This leads to much faster solutions compared to using black-box search since optimizing over the continuous search space is an easier combinatorial optimization problem that in turn leads to faster convergence. This pruning technique is not strictly consider compression in its standard definition, as it prunes during the initial training period as opposed to pruning after pretraining. This falls under the category of neural architecture search (NAS) and here they use an annealing schedule that controls the amount of pruning during NAS to incrementally make it easier to search for sub-modules that are found to have good performance in the search process. Their  $(0, \delta)$ -PAC theorem guarantees under few assumptions (see paper for further details on these assumptions) that this anneal and prune approach prunes less important weights with high probability.

### 2.2.5.2 *Sequential Monte Carlo & Reinforcement Learning Based Pruning*

**PARTICLE FILTER BASED PRUNING** Anwar, Hwang, and Sung [11] identifies important weights and paths using particle filters where the importance weight of each particle is assigned based on the misclassification rate with corresponding connectivity pattern. Particle filtering (PF) applies sequential Monte Carlo estimation with particle representing the probability density where the posterior is estimated with a random sample and parameters that are used for posterior estimation. PF propogates parameters with large magnitudes and deletes parameters with the smallest weight in re-sampling process, similar to MBP. They use PF to prune the network and retrain to compensate for the loss in performance due to PF pruning. When applied to CNNs, they reduce the size of kernel and feature map tensors while maintaining test accuracy.

**PARTICLE SWARM OPTIMIZED PRUNING** Particle Swarm Optimization (PSO) has also been combined with correlation merging algorithm (CMA) for pruning [417]. Equation 2.33 shows the PSO update formula where the velocity  $\mathbf{V}_{id}$  for  $i$ -th position of particle  $\mathbf{X}_{id}$  (i.e a parameter vector in a DNN) at the  $d$ -th iteration,

$$\mathbf{V}_{id} := \mathbf{V}_{id} + c_1 u(\mathbf{P}_{id} - \mathbf{X}_{id}) + c_2 u(\mathbf{P}_{gd} - \mathbf{X}_{id}) \quad (2.33)$$

where  $\mathbf{X}_{id} = \mathbf{X}_{id} + \mathbf{V}_{id}$ ,  $u \sim \text{Uniform}(0, 1)$  and  $c_1, c_2$  are both learning rates, corresponding to the influence social and cognition components of the swarm respectively [193]. Once the velocity vectors are updated for the DNN, the standard deviation is computed for the  $i$ -th activation as  $s_i = \sum_{p=1}^n (\mathbf{V}_{ip} - \bar{\mathbf{V}}_i)^2$  where  $\bar{v}_i$  is the mean value of  $\mathbf{V}_i$  over training samples.

Then compute Pearson correlation coefficient between the  $i$ -th and  $j$ -th unit in the hidden layer as  $\mathbf{C}_{ij} = (\mathbf{V}_{ip}\mathbf{V}_{jp} - n\bar{\mathbf{V}}_i\bar{\mathbf{V}}_j)/\mathbf{S}_i\mathbf{S}_j$  and if  $\mathbf{C}_{ij} > \tau_1$  where  $\tau$  is a predefined threshold, then merge both units, delete the  $j$ -th unit and update the weights as,

$$\mathbf{W}_{ki} = \mathbf{W}_{ki} + \alpha\mathbf{W}_{ki} \quad \text{and} \quad \mathbf{W}_{kb} = \mathbf{W}_{kb} + \beta\mathbf{W}_k \quad (2.34)$$

where,

$$\alpha = \frac{\mathbf{V}_{ip}\mathbf{V}_{jp} - n\bar{\mathbf{V}}_i\bar{\mathbf{V}}_j}{\sum_{n=1}^p \mathbf{V}_{ip}\mathbf{V}_{jp} - \bar{\mathbf{V}}_i^2}, \quad \beta = \bar{\mathbf{V}}_j - \alpha\bar{\mathbf{V}}_i \quad (2.35)$$

and  $\mathbf{W}_{ki}$  connects the last hidden layer to output unit  $k$ . If the standard deviation of unit  $i$  is less than  $\tau_2$  then it is combined with the output unit  $k$ . Finally, remove unit  $j$  and update the bias of the output unit  $k$  as  $\mathbf{W}_{kb} = \mathbf{W}_{kb} + \bar{\mathbf{V}}_i\mathbf{W}_{ki}$ . This process is repeated until a maximally compressed network that maintains performance similar to the original network is found.

**AUTOMATED PRUNING** AutoML [153] use RL to improve the efficiency of model compression performance by exploiting the fact that the sparsity of each layer is a strong signal for the overall performance. They search for a compressed architecture in a continuous space instead of searching over a discrete space. A continuous compression ratio control strategy is employed using an actor critic model (Deep Deterministic Policy Gradient [391]) which is known to be relatively stable during training, compared to alternative RL models, due to lower variance in the gradient estimator. The DDPG processes each consecutive layer, where for the  $t$ -th layer  $L_t$ , the network receives a layer embedding  $t$  that encodes information of this layer and outputs a compression ratio  $a_t$  and repeats this process from the first to last layer. The resulting pruned network is evaluated without fine-tuning, avoiding retraining to improve computational cost and time. During training, they fine-tune the best explored model given by the policy search. The MBP ratio is constrained such that the compressed model produced by the agent is below a resource constrained threshold in resource constrained case. Moreover, the maximum amount of pruning for each layer is constrained to be less than 80%. When the focus is to instead maintain accuracy, they define the reward function to incorporate accuracy and the available hardware resources.

By only requiring 1/4 number of the FLOPS they still manage to achieve a 2.7% increase in accuracy for MobileNet-V1. This also corresponds to a 1.53 times speed up on a Titan Xp GPU and 1.95 times speed up on Google Pixel 1 Android phone.

### 2.2.6 Pruning Before Training

Thus far, pruning methods has been described in the context of pruning pretrained networks. Recently, the lottery ticket hypothesis [LTH 111] showed that there exists sparse subnetworks that when trained from scratch with the same initialized weights can reach the same accuracy as the full network. The process can be formalized as the following:

1. Randomly initialize a neural network  $f(\mathbf{x}; \theta_0)$  (where  $\theta_0 \sim D_\theta$ ).
2. Train the network for  $j$  iterations, arriving at parameters  $\theta_j$
3. Prune  $p$  % of the parameters in  $\theta_j$ , creating a mask  $m$ .
4. Reset the remaining parameters to their values in  $\theta_0$ , creating the winning ticket  $f(\mathbf{x}; m \otimes \theta_0)$ .

Liu et al. [258] have further shown that the network architecture itself is more important than the remaining weights after pruning pretrained networks, suggesting pruning is better perceived as an effective architecture search. This coincides with Weight Agnostic Neural Networks [WANN; 117] search which avoids weight training. Topologies of WANNs are searched over by first sampling single shared weight for a small subnetwork and evaluated over several randomly shared weight rollout. For each rollout the cumulative reward over a trial is computed and the population of networks are ranked according to the resulting performance and network complexity. This highest ranked networks are probabilistically selected and mixed at random to form a new population. The process repeats until the desired performance and time complexity is met.

The two aforementioned findings (there exists smaller sparse subnetworks that perform well from scratch and the importance of architecture design) has revived interest in finding criteria for finding sparse and trainable subnetworks that lead to strong performance.

However, the original LTH paper was demonstrated on relatively simpler CV tasks such as MNIST and when scaled up it required careful fine-tuning of the learning rate for the lottery ticket subnetwork to achieve the same performance as the full network. To scale up LTH to larger architectures [112] in a stable way without requiring any additional fine-tuning, they relax the restrictions of reverting to the lottery ticket being found at initialization but instead revert back to the  $k$ -th epoch. This  $k$  typically corresponds to only few training epochs from initialization. Since the lottery ticket (i.e subnetwork) no longer corresponds to a randomly initialized subnetwork but instead a network trained from  $k$  epochs, they refer to these subnetworks as *matching tickets* instead. This relaxation on LTH allows tickets to be found on CIFAR-10 with ResNet-20 and ImageNet with ResNet-50, avoiding the need for using optimizer warmups to precompute learning rate statistics.

Zhou et al. [491] have further investigate the importance of the three main factors in pruning from scratch: (1) the pruning criteria used,

(2) where the model is pruned from (e.g from initialization or  $k$ -th epoch) and (3) the type of mask used. They find that the measuring the distance between the weight value at initialization and its value after training is a suitable criterion for pruning and performs at least as well as preserving weights based on the largest magnitude. They also note that if the sign is the same after training, these weights can be preserved. Lastly, they find for (3) that using a binary mask and setting weights to 0 is plays an integral part in LTH. Given that these LTH based pruning masks outperform random masks at initialization, leads to the question whether architectures can be searched for by pruning as a way of learning instead of traditional backpropogation training. In fact, [491] have also propose to use REINFORCE [405] to optimize and search for optimal wirings at each layer. The next subsection discusses recent work that aims to find optimal architectures using various criteria.

### 2.2.6.1 Pruning to Search for Optimal Architectures

Before LTH and the aforementioned line of work, Deep Rewiring [DeepR; 30] was proposed to adaptively prune and reappear periodically during training by drawing stochastic samples of network configurations from a posterior. The update rule for all active connections is given as,

$$\mathbf{W}_k \leftarrow \mathbf{W}_k - \eta \frac{\partial E}{\partial \mathbf{W}_k} - \eta \alpha + \sqrt{2\eta\Gamma} v_k \quad (2.36)$$

for  $k$ -th connection. Here,  $\eta$  is the learning rate,  $\Gamma$  is a temperature term,  $E$  is the error function and the noise  $v_k \sim \mathcal{N}(0, I\sigma^2)$  for each active weight  $\mathbf{W}$ . If the  $\mathbf{W}_k < 0$  then the connection is frozen. When the set the number of dormant weights exceeds a threshold, they reactivate dormant weights with uniform probability. The main difference between this update rule and SGD lies in the noise term  $\sqrt{2\eta\Gamma} v_k$  whereby the  $v_k$  noise and the amount of it controlled by  $\Gamma$  performs a type of random walk in the parameter space. Although unique, this approach is computationally expensive and challenging to apply to large networks and datasets.

Sparse evolutionary training [SET; 293] simplifies prune–regrowth cycles by replacing the top- $k$  lowest magnitude weights with newly randomly initialized weights and retrains and this process is repeated throughout each epoch of training. Dai, Yin, and Jha [79] carry out the same SET but using gradient magnitude as the criterion for pruning the weights. Dynamic Sparse Reparameterization [DSR; 298] implements a prune–redistribute–regrowth cycle where target sparsity levels are redistributed among layers, based on loss gradients (in contrast to SET, which uses fixed, manually configured, sparsity levels). SparseMomentum [SM; 89] follows the same cycle but instead using the mean momentum magnitude of each layer during the redistribute phase. SM outperforms DSR on ImageNet for unstructured pruning by a small margin but has no performance difference on CIFAR experiments.

Ramanujan et al. [351]<sup>2</sup> propose an edge-popup algorithm to optimize towards a pruned subnetwork from a randomly initialized network that leads to optimal accuracy. The algorithm works by switching edges until the optimal configuration is found. Each weight is assigned a “popup” score  $s_{uv}$  from neuron  $u$  to  $v$ . The top-k % percentage of weights with the highest popup score are preserved while the remaining weights are pruned. Since the top-k threshold is a step function which is non-differentiable, they propose to use a straight-through estimator to allow gradients to backpropagate and differentiate the loss with respect to  $s_{uv}$  for each respective weight i.e the activation function  $g$  is treated as the identity function in the backward pass. The scores  $s_{uv}$  are then updated via SGD. Unlike, Theis et al. [413] that use the absolute value of the gradient, they find that preserving the direction of momentum leads to better performance. During training, removed edges that are not within the top-k can switch to other positions of the same layer as the scores change. They show that this shuffling of weights to find optimal permutation leads to lower cross-entropy loss throughout training. Interestingly, this type of adaptive pruning training leads to competitive performance on ImageNet when compared to ResNet-34 and can be performed on pretrained networks.

#### 2.2.6.2 *Few-Shot and Data-Free Pruning Before Training*

Pruning from scratch requires a criterion that when applied, leads to relatively strong out-of-sample performance compared to the full network. LTH established this was possible, but the method to do so requires an intensive number of pruning-retraining steps to find this subnetwork. Recent work, has focused trying to find such subnetworks without any training, of only a few mini-batch iterations. Lee, Ajanthan, and Torr [227] aim to find these subnetworks in a single shot i.e a single pass over the training data. This is referred to as Single-shot Network Pruning (SNIP) and as in previously mentioned work it too constructs the pruning mask by measuring connection sensitivities and identifying structurally important connections.

You et al. [471] identify to as ‘early-bird’ tickets (i.e winning tickets early on in training) using a combination of early stopping, low-precision training and large learning rates. Unlike, LTH that use unstructured pruning, ‘early-bird’ tickets are identified using structured pruning whereby whole channels are pruned based on their batch normalization scaling factor. Secondly, pruning is performed iteratively within a single training epoch, unlike LTH that performs pruning after numerous retraining steps. The idea of pruning early is motivated by Saxe et al. [377] that describe training in two phase: (1) a label fitting phase where most of the connectivity patterns form and (2) a longer compression phase where the information across the networks is dispersed and lower

<sup>2</sup> This approach also is also relevant to [Section 2.2.3.2](#) as it relies on 1<sup>st</sup> order derivatives for pruning.

layers compress the input into more generalizable representations. Therefore, only phase (1) may be needed to identify important connectivity patterns and in turn find efficient sparse subnetworks. You et al. [471] conclude that this hypothesis is in fact the case when identifying channels to be pruned based on the hamming distance between consecutive pruning iterations. Intuitively, if the hamming distance is small and below a predefined threshold, channels are removed.

Tanaka et al. [410] have further investigated whether tickets can be identified without any training data. They note that the main reason for performance degradation with large amounts of pruning is due to layer collapse. Layer collapse refers when too much pruning leads to a cut-off of the gradient flow (in the extreme case, a whole layer is removed), leading to poor signal propagation and maximal compression while allowing the gradient to flow is referred to as *critical compression*.

They show that retraining with MBP avoids layer-wise collapse because gradient-based optimization encourages compression with high signal propagation. From this insight, they propose a measure for measuring synaptic flow, expressed in Equation 2.37. The parameters are first masked as  $\theta_\mu \leftarrow \mu \odot \theta_0$ . Then the iterative synaptic flow pruning objective is evaluated as,

$$\mathcal{L} = \mathbf{1}^T \left( \prod_{l=1}^T |\theta[l]_\mu| \right) \mathbf{1} \quad (2.37)$$

where  $\mathbf{1}$  is a vectors of ones. The score  $\mathcal{S}$  is then computed as  $\mathcal{S} = \frac{\partial \mathcal{L}}{\partial \theta_\mu} \odot \theta_\mu$  and the threshold  $\tau$  is defined as  $\tau = (1 - \rho - k/n)$  where  $n$  is the number of pruning iterations and  $\rho$  is the compression ratio. If  $\mathcal{S} > \tau$  then the mask  $\mu$  is updated. SynFlow achieves far higher compression ratio for the same test accuracy without requiring any data.

### 2.3 LOW RANK MATRIX & TENSOR DECOMPOSITIONS

DNNs can also be compressed by decomposing the weight tensors ( $2^{nd}$  order tensor in the case of a matrix) into a lower rank approximation which can also removed redundancies in the parameters. Many works on applying TD to DNNs have been predicated on using SVD [315, 368, 458, 459]. Hence, before discussing different TD approaches, an introduction to SVD is provided below.

A matrix  $\mathbf{A} \in \mathbb{R}^{m \times n}$  of full rank  $r$  can be decomposed as  $\mathbf{A} = \mathbf{W}\mathbf{H}$  where  $\mathbf{W} \in \mathbb{R}^{m \times r}$  and  $\mathbf{H} \in \mathbb{R}^{r \times n}$ . The change in space complexity as  $\mathcal{O}(mn) \rightarrow \mathcal{O}(r(m+n))$  at the expense of some approximation error after optimizing the following objective,

$$\min_{\mathbf{W}, \mathbf{H}} \frac{1}{2} \|\mathbf{A} - \mathbf{W}\mathbf{H}\|_F^2 \quad (2.38)$$

where for a low rank  $k < r$ ,  $\mathbf{W} \in \mathbb{R}^{m \times k}$  and  $\mathbf{H} \in \mathbb{R}^{k \times n}$  and  $\|\cdot\|_F$  is the Frobenius norm.

A common technique for achieving this low rank TD is **SVD**. For orthogonal matrices  $\mathbf{U} \in \mathbb{R}^{m \times r}$ ,  $\mathbf{V} \in \mathbb{R}^{n \times r}$  and a diagonal matrix  $\Sigma \in \mathbb{R}^{r \times r}$  of singular values,  $\mathbf{A}$  is expressed as

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T \tag{2.39}$$

where if  $k < r$  then this is called truncated **SVD**. The nonzero elements of  $\Sigma$  are the sorted in decreasing order and the top  $k$   $\Sigma_k \in \mathbb{R}^{k \times k}$  are used as  $\mathbf{A} \approx \mathbf{U}_k \Sigma_k \mathbf{V}_k^T$ .

Randomized **SVD** [139] has also been introduced for faster approximation using ideas from random matrix theory. An approximation of the range  $\mathbf{A}$  by finding  $\mathbf{Q}$  with  $r$  orthonormal columns and  $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T \mathbf{A}$ . Then the **SVD** is found by constructing a matrix  $\mathbf{B} = \mathbf{Q}^T \mathbf{A}$  and **SVD** is instead computed on  $\mathbf{B}$  as before using Equation 2.39. Since  $\mathbf{A} \approx \mathbf{Q}\mathbf{B}$ , we can see  $\mathbf{U} = \mathbf{Q}\mathbf{S}$  computes a LRD  $\mathbf{A} \approx \mathbf{U}\mathbf{S}\mathbf{V}^T$

Then as  $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^T \mathbf{A} = \mathbf{Q}(\mathbf{S}\mathbf{V}^T)$ , taking  $\mathbf{U} = \mathbf{Q}\mathbf{S}$ , leads to a low rank approximation  $\mathbf{A} \approx \mathbf{U}\mathbf{S}\mathbf{V}^T$ . Approximating  $\mathbf{Q}$  is achieved by forming a Gaussian random matrix  $\omega \in \mathbb{R}^{n \times l}$  and computing  $\mathbf{Z} = \mathbf{A}\omega$ , and using QR decomposition of  $\mathbf{Z}$ ,  $\mathbf{Q}\mathbf{R} = \mathbf{Z}$ , then  $\mathbf{Q} \in \mathbb{R}^{m \times l}$  has columns that are an orthonormal basis for the range of  $\mathbf{Z}$ .

Numerical precision is maintained by taking intermediate QR and LU decompositions during  $o$  power iterations of  $\mathbf{A}\mathbf{A}^T$  to reduce  $\mathbf{Y}$ 's spectrum because if the singular values of  $\mathbf{A}$  are  $\Sigma$ , then the singular values of  $(\mathbf{A}\mathbf{A}^T)^o$  are  $\Sigma^{2o+1}$ . With each power iteration the spectrum decays exponentially, therefore it only requires very few iterations.

### 2.3.1 Tensor Decomposition

Generalizing  $\mathbf{A}$  to higher order tensors, which can be referred to as an  $\ell$ -way array  $\mathcal{A} \in \mathbb{R}^{n_a \times n_b \dots \times n_x}$ , the aim is to find the components  $\mathcal{A} = \sum_i^r a \circ b \circ x = [[\mathbf{A}, \mathbf{B}, \dots, \mathbf{X}]]$ .

Before discussing the TD we first define three important types of matrix products used in tensor computation:

- The Kronecker product between two arbitrarily-sized matrices  $\mathbf{A} \in \mathbb{R}^{I \times J}$  and  $\mathbf{B} \in \mathbb{R}^{K \times L}$ ,  $\mathbf{A} \otimes \mathbf{B} \in \mathbb{R}^{(I) \times (JL)}$ , is a generalization of the outer product from vectors to 3 matrices  $\mathbf{A} \otimes \mathbf{B} := [\mathbf{a}_1 \otimes \mathbf{b}_1, \mathbf{a}_2 \otimes \mathbf{b}_2, \dots, \mathbf{a}_J \otimes \mathbf{b}_{L-1}, \mathbf{a}_J \otimes \mathbf{b}_L]$ .
- The Khatri-Rao product between two matrices  $\mathbf{A} \in \mathbb{R}^{I \times K}$  and  $\mathbf{B} \in \mathbb{R}^{J \times K}$ ,  $\mathbf{A}\mathbf{B} \in \mathbb{R}^{(IJ) \times K}$ , corresponds to the column-wise Kronecker product.  $\mathbf{A}\mathbf{B} := [\mathbf{a}_1 \otimes \mathbf{b}_1, \mathbf{a}_2 \otimes \mathbf{b}_2, \dots, \mathbf{a}_K \otimes \mathbf{b}_K]$ .
- The Hadamard product is the elementwise product between 2 matrices  $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{I \times J}$  and  $\mathbf{A} * \mathbf{B} \in \mathbb{R}^{I \times J}$ .

These products are used when performing Canonical Polyadic [(CP) 163], Tucker decompositions [418], Tensor Train [TT 323] to find the factor matrices  $\mathcal{X} := [[\mathbf{A}, \mathbf{B}, \dots, \mathbf{C}]]$ . For the sake of simplicity we'll proceed with 3-way tensors. As before in Equation 2.38, we can express the optimization objective as

$$\min_{\mathbf{A}, \mathbf{B}, \mathbf{C}} \sum_{i,j,k} \|\mathbf{x}_{ijk} - \sum_l \mathbf{a}_{il} \mathbf{b}_{jl} \mathbf{c}_{kl}\|^2 \quad (2.40)$$

Since the components  $\mathbf{a}, \mathbf{b}, \mathbf{c}$  are not orthogonal, we cannot compute SVD as was the case for matrices. The rank  $r$  of  $\mathbf{A}$  is also NP-hard and the solutions found for lower rank approximations may not be apart of the solution for higher ranks. Unlike, when you rotate the row or column vectors of a matrix and apply dimensionality reduction (e.g PCA) and still get the same solution, this is not the case for TD. Unlike matrices where there can be many low rank matrices, a tensor is requires to have a low-rank matrix that is compatible for all tensor slices. This interconnection between different slices results in tensor being more restrictive and hence the for weaker uniqueness conditions.

One way to perform TD using Equation 2.40 is to use alternating least squares (ALS) which involves minimizing  $\min_{\mathbf{A}}$  while fixing  $\mathbf{B}, \mathbf{C}$  and repeating this for  $\min_{\mathbf{B}}$  and  $\min_{\mathbf{C}}$ . ALS is suitable because it is a nonconvex optimization problem but with convex subproblems. CP can be generalized to different objectives apart from the squared loss, such as Rayleigh (when entries are non-negative), Boolean (entries are binary) and Poisson (when entries are counts) losses. Similar to randomized SVD described in the previous subsection, they have also been successful when scaling up TD.

With this introduction, we now move onto how low rank TD has been applied to DNNs for reducing the size of large weight matrices.

### 2.3.2 Tensor Decomposition For Attention and Recurrent Layers

In the subsection, we describe TD techniques that have been proposed for attention layers and recurrent layers.

#### 2.3.2.1 Block-Term Tensor Decomposition (BTD)

Block-Term Tensor Decomposition [(BTD) 83] combines CP decomposition and Tucker decomposition. Consider an  $n$ -th order tensor as  $\mathcal{A} \in \mathbb{R}^{A_1 \times \dots \times A_n}$  that can be decomposed into  $N$  block terms and each block consist of  $k$  elements between a core tensor  $\mathcal{G}_n \in \mathbb{R}^{G_1 \times \dots \times G_d}$  and  $d$  and factor matrices  $\mathcal{C}_n^{(k)} \in \mathbb{R}^{A_k \times G_k}$  along the  $k$ -th dimension where  $n \in [1, N]$  and  $k \in [1, d]$  [83]. BTD can then be defined as,

$$\mathcal{A} = N \sum_{n=1} \mathcal{G}_n \otimes \mathcal{C}_n^{(1)} \otimes \mathcal{C}_n^{(2)} \dots \mathcal{C}_n^{(d)} \quad (2.41)$$

The  $N$  here is the CP-rank,  $G_1, G_2, G_3$  is the Tucker-rank and  $d$  the core-order.

**BTD RNNs** Ye et al. [466] also used BTD to learn small and dense RNNs by first tensorizing the RNN weights and inputs to a 3-way tensor as  $\mathcal{X}$  and  $\mathcal{W}$  respectively. BTD is then performed on the weights  $\mathcal{W}$  and a tensorized backpropagation is computed when updating the weights. The core-order  $d$  is important when deciding the total number of parameters and they recommend  $d = [3, 5]$  which is a region that corresponds to orders of magnitude reduction in the number of parameters. When  $d > 5$  the number of parameters begins to increase again since the number of parameters is defined as  $p_{BTD} = N \sum_{k=1}^d Y_k Z_k R + R^d$  where  $Y_k$  is the row of the  $k$ -th matrix,  $Z_k$  is the number of columns and  $d$  is responsible for exponential increase in the core tensors. If  $d$  is too high, it results in the loss of spatial information. For a standard forward and backward pass, the time complexity and memory required is  $\mathcal{O}(YZ)$  for  $W$ . For BTD-RNN, the time complexity for the forward pass is  $\mathcal{O}(NdYR^d Z_{\max})$  and  $\mathcal{O}(Nd^2YR^d Z_{\max})$  on the backward pass where  $J$  is the product of the number of BT parameters for each  $d$ -th order tensor. For spatial complexity it is  $\mathcal{O}(R^d Y)$  on both passes. They find significant improvements over an LSTM baseline network and improvements over a Tensor-Train LSTM [474].

### 2.3.3 Applications of Tensor Decompositions to Convolutional Layers

This subsection now describes TD techniques proposed for structured modules, namely convolutional layers.

#### 2.3.3.1 Filter Decompositions

Rigamonti et al. [359] reduce computation in CNNs by learning a linear combination of separable filters, while maintaining performance.

For  $N$  2-d filters  $\{f^j\}_{1 \leq j \leq N}$ , one can obtain a shared set of separable (rank-1) filters by minimizing the objective in Equation 2.42

$$\arg \min_{\{f^j\}, \{m_i^j\}} \sum_i \left( \|x_i - \sum_{j=1}^N f^j * m_i^j\|_2^2 + \lambda \sum_{j=1}^N \|m_i^j\|_1 \right) \quad (2.42)$$

where  $\mathbf{x}_i$  is an input image,  $*$  denotes the convolution product operator,  $\{m_i^j\}_{j=1 \dots N}$  are the feature maps obtained during training and  $\lambda_1$  is a regularization coefficient. This can be optimized using SGD to optimize for  $m_i^j$  latent features and  $f^j$  filters.

In the first approach they identify low-rank filters using the objective in Equation 2.43 to penalize high-rank filters.

$$\operatorname{argmin}_{\{s^j\}, \{m_i^j\}} \sum_i \left( \|x_i - \sum_{j=1}^N s_j * m_j^i\|_2^2 + \lambda_1 \sum_{j=1}^N \|m_j^i\|_1 + \lambda_* \sum_{j=1}^N \|s^j\|_* \right) \quad (2.43)$$

where the  $s_j$  are the learned linear filters,  $\|*\|$  is the sum of singular values (convex relaxation of the rank), and  $\lambda_*$  is an additional regularization parameter. The second approach involves separating the optimization of squared difference between the original filter  $f^i$  and the weighted combination of learned linear filters  $w_k^j s_k$ , and the sum of singular values of learned filters  $s$ .

$$\operatorname{argmin}_{\{s_k\}, \{w_k^j\}} \sum_j \left( \|f^i - \sum_{k=1}^M w_k^j s_k\|_2^2 + \lambda_* \sum_{k=1}^M \|s^j\|_* \right) \quad (2.44)$$

They find empirically that decoupling the computation of the non-separable filters from that of the separable ones leads to better results compared to jointly optimizing over  $s^j$ ,  $m_i^j$  and  $w_k^j$  which is a difficult optimization problem.

### 2.3.3.2 Channel-wise Decompositions

Jaderberg, Vedaldi, and Zisserman [182] propose to approximate filters in convolutional layers using a low-rank basis of filters that have good separability in the spatial filter dimensions but make the contribution of removing redundancy across channels by performing channel-wise low-rank (LR) decompositions (LRD), leading to further speedups. This approach showed significant 2.5x speed ups and maintained performance on character recognition leading to SoTA on standard benchmarks.

### 2.3.3.3 Combining Filter and Channel Decompositions

Yu et al. [476] argue that sparse and low rank decompositions (LRDs) of weight filters should be combined as filters often exhibit both and ignoring either sparsity or LRDs requires iterative retraining and lower compression rates. Feature maps are reconstructed using fast-SVD. This approach allowed accuracy to be maintained for higher compression rates in few retraining steps when compared to single approaches (e.g pruning) for AlexNet, VGG-16 (15 time reduction) and LeNet CNN architectures.

## 2.4 KNOWLEDGE DISTILLATION

Knowledge distillation involves learning a smaller network from a large network using supervision from the larger network and minimizing the

entropy, distance or divergence between their logits or intermediate representations.

Bucilua, Caruana, and Niculescu-Mizil [49] were the first to explore the idea of reducing model size by learning a student network from an ensemble of models. They use a teacher network to label a large amount of unlabeled data and train a student network using supervision from the pseudo labels provided by the teacher. They find performance is close to the original ensemble with 1000 times smaller network.

Hinton, Vinyals, and Dean [162] were the first to propose distillation for a neural network, where a relatively small model (2-hidden layer with 800 hidden units and ReLU activations) is trained using supervision (class probability outputs) for the original “teacher” model (2-hidden layer, 1200 hidden units). They showed that learning from the larger network outperformed the smaller network learning from scratch in the standard supervised classification setup. In the case of learning from an ensemble of teachers, the average class probability is used as the target. The KL-divergence loss is used between the class probability outputs of the student output  $y^S$  and one-hot target  $y$  and a second term is used to ensure high fidelity between the student representation  $z^S$  and the teacher output  $z^T$ . This is expressed as,

$$\mathcal{L}_{\text{KLD}} = (1 - \alpha)\mathbb{H}(y, y^S) + \alpha\rho^2\mathbb{H}\left(\phi\left(\frac{z^T}{\rho}\right), \phi\left(\frac{z^S}{\rho}\right)\right) \quad (2.45)$$

where  $\rho$  is the temperature,  $\alpha$  balances between both terms,  $\phi$  represents the softmax function and  $\mathbb{H}(\cdot, \cdot)$  represents the entropy function. The  $\mathbb{H}\left(\phi\left(\frac{z^T}{\rho}\right), \phi\left(\frac{z^S}{\rho}\right)\right)$  is further decomposed into  $D_{\text{KL}}\left(\phi\left(\frac{z^T}{\rho}\right)\middle|\phi\left(\frac{z^S}{\rho}\right)\right)$  and a constant entropy  $\mathbb{H}\left(\phi\left(\frac{z^T}{\rho}\right)\right)$  which is not considered in learning as there are no terms depending on the student network. By using logits, as opposed to a softmax normalization across class probabilities for example, the student network better learns the relationship between each class on a log-scale which is more forgiving than the softmax when the differences in probabilities are large.

The student network is typically smaller than the teacher network and benefits from the additional information soft targets provide. There has been various extensions that involve distilling intermediate representations [360], distributions [176], maximizing mutual information between student and teacher representations [4], using pairwise interactions for improved KD [328] and contrastive representation distillation [308, 414].

#### 2.4.1 Analysis of Knowledge Distillation

The works in this subsection provide insight into the relationship between the student and teacher networks for various tasks, teacher size and network size. We also discuss work that focuses on what is required to

train a well-performing student network e.g use of early stopping [412] and avoiding training the teacher network with label smoothing [303].

**THEORIES OF WHY KNOWLEDGE DISTILLATION WORKS** For a distilled linear classifier, Phuong and Lampert [341] prove a generalization bound that shows the fast convergence of the expected loss. In the case where the number of samples is less than the dimensionality of the feature space, the weights learned by the student network are projections of the weights in the student network onto the data span. Since gradient descent makes updates that are within the data space, the student network is bounded in this space and therefore it is the best student network can approximate of the teacher network weights w.r.t the Euclidean norm. From this proof, they identify 3 important factors that explain the success of knowledge distillation:

1. The geometry of the data distribution that makes up the separation between classes greatly effects the student networks convergence rate.
2. Stochastic gradient descent is biased towards a desirable minimum in the distillation objective.
3. The loss monotonically decreases proportional to size of the training set.

**TEACHER ASSISTANT KNOWLEDGE DISTILLATION** Mirzadeh et al. [286] have shown that the performance of the student network degrades when the gap between the teacher and the student is too large for the student to learn from. Hence, they propose an intermediate ‘teaching assistant’ network to supervise and distil the student network where the intermediate networks use the logits from the teacher network. They find that teaching assisted knowledge distillation has a smoother surface around the local minima, corresponding to more robustness when the inputs are perturbed and the loss surface smoothness also corresponds to improved generalization.

**ON THE EFFICACY OF KNOWLEDGE DISTILLATION** Cho and Hariharan [64] analyse what are some of the main factors in successfully using a teacher network to distil a student network. Their main finding is that when the gap between the student and teacher networks capacity is too large, distilling a student network that maintains (or close to) performance the teacher (DenseNet or a WideResNet) is either unattainable or difficult. They also find that the student network can perform better if early stopping is used for the teacher network, as opposed to training the teacher network to convergence.

**AVOID TRAINING THE TEACHER NETWORK WITH LABEL SMOOTHING** Muller, Kornblith, and Hinton [303] show that because label

smoothing forces the same class sample representations to be closer to each other in the embedding space, it provides less information to student network about the boundary between each class and in turn leads to poorer generalization performance. They quantify the variation in logit predictions due to the hard targets using mutual information between the input and output logit and show that label smoothing reduces the mutual information. Hence, they draw a connection between label smoothing and information bottleneck principle and show through experiments that label smoothing can implicitly calibrate the predictions of a DNN.

**DISTILLING WITH NOISY LABELS** Sau and Balasubramanian [375] propose to use noise to simulate learning from multiple teacher networks by simply adding Gaussian noise to the logit outputs of the teacher network, resulting in better compression when compared to training with the original logits as targets for the teacher network. They choose a set of samples from each mini-batch with a probability  $\alpha$  to be perturbed by noise while the remaining samples are unchanged. They find that a relatively high  $\alpha = 0.8$  performed the best for image classification task, corresponding to 80% of teacher logits having noise.

Li et al. [237] distill models with noisy labels and use a small dataset with clean labels, alongside a knowledge graph that contains the label relations, to estimate risk associated with training using each noisy label. A model is trained on the clean dataset  $D_c$  and the main model is trained over the whole dataset  $D$  with noisy labels using the loss function,

$$\mathcal{L}_D(\mathbf{y}_i, f(\mathbf{x}_i)) = \lambda l(\mathbf{y}_i, f(\mathbf{x}_i)) + (1 - \lambda)l(\mathbf{s}_i, f(\mathbf{x}_i)) \quad (2.46)$$

where  $\mathbf{s}_i = \delta[f_c^D(\mathbf{x}_i)]$ . The first loss term is cross entropy between student noisy and noisy labels and the second term is the loss between the hard target  $\mathbf{s}_i$  given by the model trained on clean data and the model trained on noisy data.

They also use pseudo labels  $\hat{\mathbf{y}}\lambda_i = \lambda\mathbf{y}_i + (1 - \lambda)\mathbf{s}_i$  that combine noisy label  $\mathbf{y}_i$  with the output  $\mathbf{s}_i$  trained on  $D_c$ . This is motivated by the fact that both noisy label and the predicted labels from clean data are independent and this can be closer to true labels  $\mathbf{y}_i^*$  under conditions which they further detail in the paper. To avoid the model trained on  $D_c$  overfitting, they assign label confidence score based on related labels from a knowledge graph, resulting in a reduction in model variance during knowledge distillation.

#### 2.4.1.1 Distillation of Hidden Layer Activation Boundaries

Instead of transferring the outputs of the teacher network, Heo et al. [157] transfer activation boundaries, essentially outputs which neurons are activated and those that are not. They use an activation loss

that minimizes the difference between the student and teacher network activation boundaries, unlike previous work that focuses on the activation magnitude. Since gradient descent updates cannot be used on the non-differentiable loss, they propose an approximation of the activation transfer loss that can be minimized using gradient descent. The objective is given as,

$$\begin{aligned} \mathcal{L}(I) = & \|\rho(\mathcal{T}(I))\sigma(\mu\mathbf{1} - r(\mathcal{S}(I))) + \\ & (1 - \rho(\mathcal{T}(I))) \circ \sigma(\mu\mathbf{1} + r(\mathcal{S}(I)))\|_2^2 \end{aligned} \quad (2.47)$$

where  $\mathcal{S}(I)$  and  $\mathcal{T}(I)$  are the neuron response tensors for student and teacher networks,  $\rho(\mathcal{T}(I))$  is the the activation of teacher neurons corresponding to class labels,  $r(\mathcal{S}(I))$  is the ,  $r$  is a connector function (a fully connected layer in their experiments) that converts a neuron response vector of student to the same size as the teacher vector,  $\circ$  is elementwise product of vectors and  $\mu$  is the margin to stabilize training.

**SIMULATING ENSEMBLED TEACHERS TRAINING** Park et al. [325] have extended the idea of student network learning from a noisy teacher to speech recognition and similarly found high compression rates. Han et al. [141] have pointed out that co-teaching (where two networks learn from each other where one has clean outputs and the other has noisy outputs) avoids a single DNN from learning to memorize the noisy labels and select samples from each mini-batch that the networks should learn from and avoid those samples which correspond to noisy labels. Since both networks have different ways of learning, they filter different types of error occurring from the noisy labels and this information is communicated mutually. This strategy could also be useful for using the teacher network to provide samples to a smaller student network that improve the learning of the student.

#### 2.4.2 *Data-Free Knowledge Distillation*

Lopes, Fenu, and Starner [262] aim to distill in the scenario where it is not possible to have access to the original data the teacher network was trained on. This can occur due to privacy issues (e.g personal medical data, models trained case-based legal data) or the data is no longer available or some way corrupted. They store the sufficient statistics (e.g mean and covariance) of activation outputs from the original data along with the pretrained teacher network to reconstruct the original training data input. This is achieved by trying to find images that have the highest representational similarity to those given by the representations from the activation records of the teacher network. Gaussian noise is passed as input to the teacher and update gradients to the noise to minimize the difference between the recorded activation outputs and

those of the noisy image and repeat this to reconstruct the teachers view of the original data.

They manage to compress the teacher network to half the size in the student network using the reconstructed inputs constructed from using the metadata. The amount of compression achieved is contingent on the quality of the metadata, in their case they only used activation statistics. We posit that the notion of creating synthetic data from summary statistics of the original data to train the student network is worth further investigation.

### 2.4.3 *Distilling Recurrent (Autoregressive) Neural Networks*

Although the work by Bucilua, Caruana, and Niculescu-Mizil [49] and Hinton, Vinyals, and Dean [162] has often proven successful for reducing the size of neural models in other non-sequential tasks, many sequential tasks in NLP and CV have high-dimensional outputs (machine translation, pixel generation, image captioning etc.). This means using the teachers probabilistic outputs as targets can be expensive.

Kim and Rush [201] use the teachers hard targets (also 1-hot vectors) given by the highest scoring beam search prediction from an encoder-decoder RNN, instead of the soft output probability distribution. The teacher distribution  $q(y_t|x)$  is approximated by its mode:  $q(y_s|x) \approx 1t = \arg \max_{y_t \in \mathcal{Y}} q(y_t|x)$  with the following objective:

$$\begin{aligned} \mathcal{L}_{SEQ-MD} &= -\mathbb{E}_{x \sim D} \sum_{y_t \in \mathcal{Y}} p(y_t|x) \log p(y_t|x) \approx \\ &-\mathbb{E}_{x \sim D}, \hat{y}_s = \arg \max_{y_t \in \mathcal{Y}} q(y_t|x) [\log p(y_t = \hat{y}_s|x)] \end{aligned} \quad (2.48)$$

where  $y_t \in \mathcal{Y}$  are teacher targets (originally defined by the predictions with the highest scoring beam search) in the space of possible target sequences. When the temperature  $\tau \rightarrow 0$ , this is equivalent to standard knowledge distillation.

In sequence-level interpolation, the targets from the teacher with the highest *similarity* with the ground truth are used as the targets for the student network. Experiments on NMT showed performance improvements compared to soft targets and further pruning the distilled model results in a pruned student that has 13 times fewer parameters than the teacher network with a 0.4 decrease in BLEU metric.

### 2.4.4 *Distilling Transformer-based (Non-Autoregressive) Networks*

Knowledge distillation has also been applied to very large transformer networks, predominantly on BERT [90] given its wide success in NLP. Thus, there has been a lot of recent work towards reducing the size of BERT and related models using knowledge distillation.

**DISTILBERT** Sanh et al. [373] achieves distillation by training a smaller BERT on very large batches using gradient accumulation, uses dynamic masking, initializes the student weights with teacher weights and removes the next sentence prediction objective. They train the smaller BERT model on the original data BERT was trained on and find that DistilBERT is within 3% of the original BERT accuracy while being 60% faster when evaluated on the GLUE [432] benchmark dataset.

**BERT PATIENT KNOWLEDGE DISTILLATION** Instead of minimizing the soft probabilities between the student and teacher network outputs, Sun et al. [402] propose to also learn from the intermediate layers of the BERT teacher network by minimizing the mean squared error between adjacent and normalized hidden states. This loss is combined with the original objective proposed by Hinton, Vinyals, and Dean [162] which showed further improves in distilling BERT on the GLUE benchmark datasets [432].

**TINYBERT** TinyBERT [186] combines multiple Mean Squared Error (MSE) losses between embeddings, hidden layers, attention layers and prediction outputs between  $S$  and  $T$ . The TinyBERT distillation objective is shown below, where it combines multiple reconstruction errors between  $S$  and  $T$  embeddings (when  $m = 0$ ), between the hidden and attention layers of  $S$  and  $T$  when  $M \geq m > 0$  where  $M$  is index of the last hidden layer before prediction layer and lastly the cross entropy between the predictions where  $t$  is the temperature of the softmax.

Through many ablations in experimentations, they find distilling the knowledge from multi-head attention layers to be an important step in improving distillation performance.

**ALBERT** Lan et al. [217] proposed factorized embeddings to reduce the size of the vocabulary embeddings and weight sharing across layers to reduce the number of parameters without a performance drop and further improve performance by replacing next sentence prediction with an inter-sentence coherence loss. ALBERT is 5.5% the size of original BERT and has produced state of the art results on top NLP benchmarks such as GLUE [432], SQuAD [350] and RACE [213].

**BERT DISTILLATION FOR TEXT GENERATION** Chen et al. [63] use a conditional masked language model that enables BERT to be used on generation tasks. The outputs of a pretrained BERT teacher network are used to provide sequence-level supervision to improve Seq2Seq model and allow them to plan ahead.

**APPLICATIONS TO MACHINE TRANSLATION** Zhou, Neubig, and Gu [490] seek to better understand why knowledge distillation leads to better non-autoregressive distilled models for machine translation.

They find that the student network finds it easier to model variations in the output data since the teacher network reduces the complexity of the dataset.

#### 2.4.5 Ensemble-based Knowledge Distillation

ENSEMBLES OF TEACHER NETWORKS FOR SPEECH RECOGNITION Chebotar and Waters [57] use the labels from an ensemble of teacher networks to supervise a student network trained for acoustic modelling. To choose a good ensemble, one can select an ensemble where each individual model potentially make different errors but together they provide the student with strong signal for learning. Boosting weights each sample based proportional to its misclassification rate. Similarly this can be used on the ensemble to learn which outputs from each model to use for supervision. Instead of learning from a combination of teachers that are best by using an oracle that approximates the best outcome of the ensemble for automatic speech recognition (ASR) as

$$P_{\text{oracle}}(s|x) = \sum_{i=1}^N [O(u) = i] P_i(s|x) = P_{O(u)}(s|x) \quad (2.49)$$

where the oracle  $O(u) \in 1 \dots N$  that contains  $N$  teachers assigns all the weight to the model that has the lowest word errors for a given utterance  $u$ . Each model is an RNN of different architecture trained with different objectives and the student  $s$  is trained using the Kullback Leibler (KL) divergence between oracle assigned teachers output and the student network output. They achieve an 8.9% word error rate improvement over similarly structured baseline models.

Freitag, Al-Onaizan, and Sankaran [113] apply knowledge distillation to NMT by distilling an ensemble of networks and oracle BLEU teacher network into a single NMT system. They find a student network of equal size to the teacher network outperforms the teacher after training. They also reduce training time by only updating the student networks with filtered samples based on the knowledge of the teacher network which further improves translation performance.

Cui et al. [75] propose two strategies for learning from an ensemble of teacher network; (1) alternate between each teacher in the ensemble when assigning labels for each mini-batch and (2) simultaneously learn from multiple teacher distributions via data augmentation. They experiment on both approaches where the teacher networks are deep VGG and LSTM networks from acoustic models.

Cui et al. [75] extend knowledge distillation to multilingual problems. They use multiple pretrained teacher LSTMs trained on multiple low-resource languages to distil into a smaller standard (fully-connected) DNN. They find that student networks with good input features makes it easier to learn from the teachers labels and can improve over the

original teacher network. Moreover, from their experiments they suggest that allowing the ensemble of teachers learn from one another, the distilled model further improves.

**MEAN TEACHER NETWORKS** Tarvainen and Valpola [412] find that averaging the models weights of an ensemble at each epoch is more effective than averaging label predictions for semi-supervised learning. This means the Mean Teacher can be used as unsupervised learning distillation approach as the distiller does not need labels. than methods which rely on supervision for each ensemble model. They find this straightforward approach to outperform previous ensemble based distillation approaches [215] when only given 1000 labels on the Street View House View Number [SVHN; 127] dataset. Moreover, using Mean Teacher networks with Residual Networks achieved SoTA with 4000 labels from 10.55% error to 6.28% error.

**ON-THE-FLY NATIVE ENSEMBLE** Zhu, Gong, et al. [495] focus on using distillation on the fly in a scenario where the teacher may not be fully pretrained or it does not have a high capacity. This reduces compression from a two-phase (pretrain then distil) to one phase where both student and teacher network learn together. They propose an On the fly Native Ensemble (ONE) learning strategy that essentially learns a strong teacher network that assists the student network as it is learning. Performance improvements for on the fly distillation are found on the top benchmark image classification datasets.

**MULTI-TASK TEACHER NETWORKS** Liu et al. [254] perform knowledge distillation for performing multi-task learning (MTL), using the outputs of teacher models from each natural language understanding (NLU) task as supervision for the student network to perform MTL. The distilled MT-DNN outperforms the original network on 7 out of 9 NLU tasks (includes sentence classification, pairwise sentence classification and pairwise ranking) on the GLUE [432] benchmark dataset.

#### 2.4.5.1 *Meta-Embeddings*

Meta-embedding (ME) learning is an emerging approach has been used to learn more accurate word embeddings given existing (source) word embeddings as the sole input. Due to their ability to incorporate semantics from multiple source embeddings in a compact manner with superior performance, ME learning has gained popularity among practitioners in NLP. Given independently trained multiple word representations (aka *embeddings*) learnt using diverse algorithms and lexical resources, word ME (ME) learning methods [22, 39, 66, 149, 184, 451, 469] attempt to learn more accurate and wide-coverage word embeddings. The input and output word embeddings to the ME algorithm are referred respectively as the *source* and *meta*-embeddings.

The source embeddings used as the inputs to an ME learning method can be trained using different methods. For example, context-insensitive *static* word embedding methods [67, 92, 173, 281, 289, 335] represent a word by a single vector that does not vary depending on the context in which the word occurs. On the other hand, *contextualised* word embedding methods [90, 217, 255, 336, 463] represent the same word with different embeddings in its different contexts. It is not clear beforehand which word embedding is best for a particular NLP task. By being agnostic to the underlying differences in the source embedding learning methods, ME learning methods are in principle able to incorporate a wide range of source word embeddings. Moreover, this decoupling of source embedding learning from the subsequent ME learning process simplifies the latter.

ME learning methods must not assume the access to the original training resources used to train the source word embeddings. Source word embeddings can be trained using different linguistic resources such as text corpora and dictionaries [415]. Although pretrained word embeddings are often publicly released and are free of charge to use, the resources on which those embeddings were originally trained might not be publicly available due to various constraints such as copyright and licensing restrictions. Consequently, ME learning methods have not assumed the access to the original training resources that were used to train the source embeddings during the ME learning process. In other words, all semantic information used by an ME learning method must be obtained only from the source word embeddings.

ME learning methods must be able to handle pretrained word embeddings of different dimensionalities. Because the ME learning methods operate directly on pretrained source word embeddings without retraining them on linguistic resources, the dimensionalities of the source word embeddings are often different. Prior work [231, 470] studying word embeddings have showed that the performance of a static word embedding is directly influenced by its dimensionality. ME learning methods use different techniques such as concatenation [469], orthogonal projections [149, 184] and averaging [66] after applying zero-padding to the sources with smaller dimensionalities as necessary to handle source embeddings with different dimensionalities.

ME learning is attractive from an NLP practitioners point for several reasons. First, as mentioned above, there is already a large number of pretrained and publicly available repositories of static and contextualised word embeddings. However, it is not readily obvious what is the best word embedding method to represent the input in a particular NLP application. We might not be able to try each one of those embeddings due to both time and computational constraints. ME learning provides a convenient alternative to selecting the single best word embedding, where we can use *all* source word embeddings to create a single ME and use that for representing input texts in an NLP application. Second, *unsupervised*

*ME learning methods* (Section 2.4.6) do *not* require labelled data when creating an ME from a given set of source word embeddings. This is particularly attractive in scenarios where we do not have sufficiently large training resources for learning word embeddings from scratch but have access to multiple pretrained word embeddings. Moreover, by using multiple source embeddings we might be able to overcome the limited vocabulary coverage in the individual sources. Third, in situations where there is some labelled data for the target task or domain, we can use *supervised ME learning methods* (Equation 2.4.6) to fine-tune the MEs for that task or domain.

From a theoretical point-of-view, ME learning can be seen as an instance of ensemble learning [7, 345], where we incorporate information from multiple models of lexical (word-level) semantics to learn an improved representation model. An ensemble typically helps to cancel out noise in individual models, while reinforcing the useful patterns repeated in multiple models [304]. Although there are some theoretical work studying word embedding learning [13, 300], the theoretical analysis of ME learning remains an open research topic. For example, under what conditions can we learn a better ME than individual source embeddings is an important theoretical consideration. ME learning can also be seen as related to *model distillation* [10, 162] where we must learn a simpler *student* model from a more complicated *teacher* model. Model distillation is an actively researched topic in deep learning where it is attractive to learn smaller networks involving a lesser number of parameters from a larger network to avoid overfitting and inference-time efficiency.

#### 2.4.6 Meta-Embedding Problem Definition

Let us consider a set of  $N$  source word embeddings  $s_1, s_2, \dots, s_N$  respectively covering vocabularies (i.e. sets of words)  $\mathcal{V}_1, \mathcal{V}_2, \dots, \mathcal{V}_N$ . The embedding of a word  $w$  in  $s_j$  is denoted by  $\mathbf{s}_j(w) \in \mathbb{R}^{d_j}$ , where  $d_j$  is the dimensionality of  $s_j$ . We can represent  $s_j$  by an embedding matrix  $\mathbf{E}_j \in \mathbb{R}^{d_j \times |\mathcal{V}_j|}$ . For example,  $\mathbf{E}_1$  could be the embedding matrix obtained by running skip-gram with negative sampling [(SGNS) 281] on a corpus, whereas  $\mathbf{E}_2$  could be that obtained from global vector prediction [(GloVe) 335] etc. Then, the problem of ME can be stated as – *what is the optimal way to combine  $\mathbf{E}_1, \dots, \mathbf{E}_n$  such that some goodness measure defined for the accuracy of the semantic representation for the words is maximised?*

The source word embeddings in general do not have to cover the same set of words. If  $w \notin \mathcal{V}_n$ , we can either assign a zero embedding or a random embedding as  $\mathbf{s}_n(w)$  as a workaround. [469] proposed a method to predict source embeddings for the words missing in a particular source. Specifically, given two different sources  $s_n$  and  $s_m$  (where  $n \neq m$ ) they learn a projection matrix  $\mathbf{A} \in \mathbb{R}^{d_m \times d_n}$  using the

words in  $\mathcal{V}_n \cap \mathcal{V}_m$ , the intersection between the vocabularies covered by both  $s_n$  and  $s_m$ . They find  $\mathbf{A}$  by minimising the sum of squared loss,  $\sum_{w \in \mathcal{V}_n \cap \mathcal{V}_m} \|\mathbf{A}s_n(w) - s_m(w)\|_2^2$ . Finally, we can predict the source embedding for a word  $w' \notin \mathcal{V}_m$  and  $w' \in \mathcal{V}_n$  using the learnt  $\mathbf{A}$  as  $\mathbf{A}s_n(w')$ . If we have multiple sources, we can learn such projection matrices between each pair of sources and in both directions. We can then, for example, consider the average of all predicted embeddings for a word as its source embedding in a particular source. After this preprocessing step, all words will be covered by all source embeddings. [469] showed that by applying this preprocessing step prior to learning MEs (referred to as the 1TON+ method in their paper) to significantly improve the performance of the learnt MEs. However, as we see later, much prior work in ME learning do assume a common vocabulary over all source embeddings for simplicity. Without loss of generality, we will assume that all words are covered by a common vocabulary  $\mathcal{V}$  after applying any one of the above-mentioned methods.

**UNSUPERVISED META-EMBEDDING LEARNING** In unsupervised ME we do not assume the availability of any manually-annotated labelled data that we can use in the learning process. In this setting all data that we have at our disposal is limited to the pretrained source embeddings.

**CONCATENATION** One of the simplest approaches to create an ME under the unsupervised setting is vector concatenation [22, 39, 469]. Denoting concatenation by  $\oplus$ , we can express the concatenated ME,  $\mathbf{m}_{\text{conc}}(w) \in \mathbb{R}^{d_1 + \dots + d_N}$ , of a word  $w \in \mathcal{V}$  by Equation (2.50).

$$\mathbf{m}_{\text{conc}}(w) = \mathbf{s}_1(w) \oplus \dots \oplus \mathbf{s}_N(w) = \bigoplus_{j=1}^N \mathbf{s}_j(w) \quad (2.50)$$

Goikoetxea, Agirre, and Soroa [124] showed that the concatenation of word embeddings learnt separately from a corpus and the WordNet to produce superior word embeddings.

Yin and Schütze [469] post-processed the MEs created by concatenating the source embeddings using SVD to reduce the dimensionality. However, applying SVD often results in degradation of accuracy in the MEs compared to the original concatenated version [39].

It is easier to see that concatenation does not remove any information that is already covered by the source embeddings. However, it is not obvious under what conditions concatenation could produce an ME that is superior to the input source embeddings.

**AVERAGING** Note that as we already mentioned, source embeddings are trained independently and can have different dimensionalities. Even when the dimensionalities do agree, vectors that lie in different vector spaces cannot be readily averaged. However, rather surprisingly, [66] showed that accurate MEs can be produced by first zero-padding source

embeddings as necessary to bring them to a common dimensionality and then by averaging to create  $\mathbf{m}_{\text{avg}}(w)$  as given by (2.51).

$$\mathbf{m}_{\text{avg}}(w) = \frac{1}{N} \sum_{j=1}^N \mathbf{s}_j^*(w) \quad (2.51)$$

Here,  $\mathbf{s}_j^*(w)$  is the zero-padded version of  $\mathbf{s}_j(w)$  such that its dimensionality is equal to  $\max(d_1, \dots, d_N)$ . In contrast to concatenation, averaging has the desirable property that the dimensionality of the ME is upper-bounded by  $\max(d_1, \dots, d_N) < \sum_{j=1}^N d_j$ . [66] showed that when word embeddings in each source are approximately orthogonal, a condition that they empirically validate for pre-trained word embeddings, averaging can approximate the MEs created by concatenating.

Although averaging does not increase the dimensionality of the ME space as with concatenation, it does not consistently outperform concatenation, especially when the orthogonality condition does not hold. To overcome this problem, [184] proposed to first learn orthogonal projection matrices for each source embedding space. They measure the Mahalanobis metric between the projected source embeddings, which is a generalisation of the inner-product that does not assume the dimensions in the vector space to be uncorrelated.

To explain their proposal further, let us consider two sources  $s_1$  and  $s_2$  with identical dimensionality  $d$ . Let us assume the orthogonal projection matrices for  $s_1$  and  $s_2$  to be respectively  $\mathbf{A}_1 \in \mathbb{R}^{d \times d}$  and  $\mathbf{A}_2 \in \mathbb{R}^{d \times d}$ . The two words  $w_i, w_j \in \mathcal{V}_1 \cap \mathcal{V}_2$  are projected to a common space respectively as  $\mathbf{A}_1 \mathbf{s}_1(w_i) \in \mathbb{R}^d$  and  $\mathbf{A}_2 \mathbf{s}_2(w_j) \in \mathbb{R}^d$ . The similarity in this projected space is computed using a Mahalanobis metric  $(\mathbf{A}_1 \mathbf{s}_1(w_i))^\top \mathbf{B} (\mathbf{A}_2 \mathbf{s}_2(w_j))$  defined by the matrix  $\mathbf{B} \in \mathbb{R}^{d \times d}$ . They learn  $\mathbf{A}_1, \mathbf{A}_2$  and  $\mathbf{B}$  such that the above metric computed between the projected source embeddings of the same word is close 1, while that for two different words is close to 0. Their training objective can be written concisely as in (2.52) using the embedding matrices  $\mathbf{E}_1, \mathbf{E}_2 \in \mathbb{R}^{d \times |\mathcal{V}_1 \cap \mathcal{V}_2|}$  and a matrix  $\mathbf{Y}$  where the  $(i, j)$  element  $Y_{ij} = 1$  if  $w_i = w_j$  and 0 otherwise.

$$\min_{\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}} \left\| \mathbf{E}_1^\top \mathbf{A}_1^\top \mathbf{B} \mathbf{A}_2 \mathbf{E}_2 - \mathbf{Y} \right\|_F^2 + \lambda \|\mathbf{B}\|_F^2 \quad (2.52)$$

Here,  $\lambda \geq 0$  is a regularisation coefficient corresponding to the Frobenius norm regularisation of  $\mathbf{B}$ , which prefers smaller Mahalanobis matrices. They show that the averaging of the projected source embeddings (i.e.  $(\mathbf{B}^{\frac{1}{2}} \mathbf{A}_1 \mathbf{s}_1(w) + \mathbf{B}^{\frac{1}{2}} \mathbf{A}_2 \mathbf{s}_2(w))/2$ ) to outperform simple non-projected averaging (given by (2.51)). Learning such orthogonal projections for the sources has shown to be useful even in supervised ME learning [149] as discussed later in Equation 2.4.6.

**LINEAR PROJECTIONS** In their pioneering work on ME, [469] proposed to project source embeddings to a common space via source-specific linear transformations, which they refer to as 1TON. They require that the ME of a word  $w$ ,  $\mathbf{m}_{1\text{TON}}(w) \in \mathbb{R}^{d_m}$ , reconstruct each source embedding,  $\mathbf{s}_j(w)$  of  $w$  using a linear projection matrix,  $\mathbf{A}_j \in \mathbb{R}^{d_j \times d_m}$ , from  $s_j$  to the ME space by as given by (2.53).

$$\hat{\mathbf{s}}_j(w) = \mathbf{A}_j \mathbf{m}_{1\text{TON}}(w) \quad (2.53)$$

Here,  $\hat{\mathbf{s}}_j(w)$  is the reconstructed source embedding of  $w$  from the ME. Next, the squared Euclidean distance between the source- and MEs is minimised over all words in the intersection of the source vocabularies, subjected to Frobenius norm regularisation as in (2.54).

$$\min_{\substack{\forall_{j=1}^N \mathbf{A}_j \\ \forall_{w \in \mathcal{V}} \mathbf{m}_{1\text{TON}}(w)}} \sum_{j=1}^N \alpha_j \left( \sum_{w \in \mathcal{V}} \|\hat{\mathbf{s}}_j(w) - \mathbf{s}_j(w)\|_2^2 + \|\mathbf{A}_j\|_F^2 \right) \quad (2.54)$$

They use different weighting coefficients  $\alpha_j$  to account for the differences in accuracies of the sources. They determine  $\alpha_j$  using the Pearson correlation coefficients computed between the human similarity ratings and cosine similarity computed using the each source embedding between word pairs on the [285] dataset. The parameters can be learnt using SGD, alternating between projection matrices and MEs.

[304] showed that by requiring the projection matrices to be orthogonal (corresponding to the Orthogonal Procrustes Problem) the accuracy of the learnt MEs is further improved. However, 1TON requires all words to be represented in all sources. To overcome this limitation they predict the source embedding for missing words as described in Section 2.4.6.

Assuming that a single *global* linear projection can be learnt between the ME space and each source embedding as done by [469] is a stronger requirement. [39] relaxed this requirement by learning *locally linear* (LLE) MEs. To explain this method further let us consider computing the LLE-based ME,  $\mathbf{m}_{\text{LLE}}(w)$ , of a word  $w \in \mathcal{V}_1 \cap \mathcal{V}_2$  using two sources  $s_1$  and  $s_2$ . First, they compute the set of nearest neighbours,  $\mathcal{N}_j(w)$ , of  $w$  in  $s_j$  and represent  $w$  as the linearly-weighted combination of its neighbours by a matrix  $\mathbf{A}$  by minimising (2.55).

$$\min_{\mathbf{A}} \sum_{j=1}^2 \sum_{w \in \mathcal{V}_1 \cap \mathcal{V}_2} \left\| \mathbf{s}_j(w) - \sum_{w' \in \mathcal{N}_j(w)} A_{ww'} \mathbf{s}_j(w') \right\|_2^2 \quad (2.55)$$

They use AdaGrad to find the optimal  $\mathbf{A}$ . Next, MEs are learnt by minimising (2.56) using the learnt neighbourhood reconstruction weights in  $\mathbf{A}$  are preserved in a vector space common to all source embeddings.

$$\sum_{w \in \mathcal{V}_1 \cap \mathcal{V}_2} \left\| \mathbf{m}_{\text{LLE}}(w) - \sum_{j=1}^2 \sum_{w' \in \mathcal{N}_j(w)} C_{ww'} \mathbf{m}_{\text{LLE}}(w') \right\|_2^2 \quad (2.56)$$

Here,  $C_{ww'} = A_{ww'} \sum_{j=1}^2 \mathbb{I}[w' \in \mathcal{N}_j(w)]$ , where  $\mathbb{I}$  is the indicator function which returns 1 if the statement evaluated is True. Optimal MEs can then be found by solving an eigendecomposition of the matrix  $(\mathbf{I} - \mathbf{C})^\top (\mathbf{I} - \mathbf{C})$ , where  $\mathbf{C}$  is the matrix formed by arranging  $C_{ww'}$  as the  $(w, w')$  element. This approach has the advantage that it does not require all words to be represented by all sources, thereby obviating the need to predict missing source embeddings prior to ME.

**AUTOENCODING** [22] modelled ME learning as an *autoencoding* problem where information embedded in different sources are integrated at different levels to propose three types of MEs: Decoupled Autoencoded ME (DAEME) (independently encode each source and concatenate), Concatenated Autoencoded ME (CAEME) (independently decode MEs to reconstruct each source), and Averaged Autoencoded ME (AAEME) (similar to DAEME but instead of concatenation uses averaging). Given the space constraints we describe only the AAEME model, which was the overall best performing among the three.

Consider two sources  $s_1$  and  $s_2$ , which are encoded respectively by two encoders  $E_1$  and  $E_2$ . The AAEME of  $w$  is computed as the  $\ell_2$  normalised average of the encoded source embeddings as given by (2.57).

$$\mathbf{m}_{\text{AAEME}}(w) = \frac{E_1(\mathbf{s}_1(w)) + E_2(\mathbf{s}_2(w))}{\|E_1(\mathbf{s}_1(w)) + E_2(\mathbf{s}_2(w))\|_2} \quad (2.57)$$

Two independent decoders,  $D_1$  and  $D_2$ , are trained to reconstruct the two sources from the ME.  $E_1, E_2, D_1$  and  $D_2$  are jointly learnt to minimise the a weighted reconstruction loss.

In comparison to methods that learn globally or locally linear transformations [39, 469], autoencoders learn nonlinear transformations. Their proposed autoencoder variants outperform 1TON and 1TON+ on multiple benchmark tasks.

**SUPERVISED META-EMBEDDING LEARNING** MEs have also been learned specifically for a set of supervised tasks. Unlike unsupervised ME learning methods described in Section 2.4.6, supervised MEs use end-to-end learning and fine-tune the MEs specifically for the downstream tasks of interest.

[307] used MEs to regularise a supervised learner by reconstructing the ensemble set of pretrained embeddings as an auxiliary task to the main supervised task whereby the encoder is shared between both tasks. (2.58) shows the auxiliary reconstruction mean squared error loss weighted by  $\alpha$  for each word  $w_t$  in a sequence of of length  $T$  words,

and the main sequence classification task cross-entropy loss, weighted by  $\beta$ . Here,  $f_{\theta_{\text{aux}}}$  is the subnetwork of  $f_{\theta}$  that corresponds to the ME reconstruction and  $f_{\theta_{\text{main}}}$  is the subnetwork used to learn on the main task, i.e.  $f_{\theta}$  except the decoder layer of the ME autoencoder.

$$\min \frac{1}{TN} \sum_{t=1}^T \left[ \alpha \left( \mathbf{f}_{\theta_{\text{aux}}}(\mathbf{m}_{\text{conc}}(w_t)) - \mathbf{m}_{\text{conc}}(w_t) \right)^2 + \beta \sum_{c=1}^C \mathbf{f}_{\theta_{\text{main}}}(\mathbf{m}_{\text{conc}}(w_t)) \log y_{t,c} \right] \quad (2.58)$$

The ME reconstruction shows improved performance on both intrinsic tasks (word similarity and relatedness) and extrinsic tasks (named entity recognition, part of speech tagging and sentiment analysis). They also show that MEs require less labelled data for Universal Part of Speech tagging<sup>3</sup> to perform as well as unsupervised MEs. [451] too successfully deploy the aforementioned ME regularisation in the supervised learning setting. They showed that as the ME hidden layer becomes smaller, the improvement in performance for supervised MEs over unsupervised MEs becomes larger.

[196] proposed a supervised ME learning method where they compute the ME of a word using a dynamically-weighted sum of the projected source word embeddings. Each source embedding,  $s_j$ , is projected to a common  $d$ -dimensional space using a matrix  $\mathbf{P}_j \in \mathbb{R}^{d_j \times d}$  and a basis vector  $\mathbf{b}_j \in \mathbb{R}^d$  as given by (2.59).

$$\mathbf{s}'_j(w) = \mathbf{P}_j \mathbf{s}_j(w) + \mathbf{b}_j \quad (2.59)$$

Next, the Dynamic Meta Embedding,  $\mathbf{m}_{\text{dme}}(w)$ , of a word  $w$  is computed as  $\mathbf{m}_{\text{dme}}(w) = \sum_{j=1}^N \alpha_{w,j} \mathbf{s}'_j(w)$ . Here,  $\alpha_{w,j}$  is the scalar weight associated with  $w$  in source  $s_j$ , and is computed via self-attention mechanism as given by  $\alpha_{w,j} = \phi(\mathbf{a}^\top \mathbf{s}'_j(w) + b)$ . Here,  $\phi$  is an activation function such as softmax,  $\mathbf{a} \in \mathbb{R}^d$  and  $b \in \mathbb{R}$  are respectively source and word independent attention parameters to be learnt using labelled data. They also proposed a contextualised version of DME (CDME), where they used the hidden state from a BiLSM that takes the projected source embeddings as the input. Although the differences between DME and CDME were not statistically significant, overall, the highest maximum performances were reported with CDME.

[456] extended this approach by introducing task-specific factors for a downstream task by computing the pairwise interactions between embeddings in the ensemble set. [149] also create task-specific MEs by learning orthogonal transformations to linearly combine the source embeddings. As already discussed in Equation 2.4.6, enforcing orthogonality on the transformation matrix has shown to improve performance of ME.

<sup>3</sup> <https://universaldependencies.org/u/pos/all.html>

SENTENCE-LEVEL META-EMBEDDING LEARNING Our focus in this survey paper is word-level ME learning. However, a closely-related extension of this is sentence-level ME. [344] proposed several methods to combine sentence-embeddings from pretrained encoders such as by concatenating and averaging the individual sentence embeddings. These methods correspond to using sentence embeddings instead of source word embeddings in (2.50) and (2.51) with  $\ell_2$  normalised sources. They also used SVD on a matrix formed by concatenated sentence embeddings for a set of sentences to obtain lower-dimensional sentence-level MEs.

Canonical Correlation Analysis (CCA) is a method to find optimal linear projections to two vector spaces such that they are maximally correlated. [344] used the Generalised CCA (GCCA), which extends CCA to more than three random vectors, to learn sentence-level MEs. They also extend AAEME method described in Equation 2.4.6 to multiple sentence encoders, where they learn an autoencoder between each pair of sources. The reconstruction losses for all encodes and decoders are jointly minimised and the average of the encoded sentences is taken as the ME of a sentence.

MULTI-LINGUAL META-EMBEDDING LEARNING MEs have also been extended to the cross-lingual and multi-lingual settings. Winata, Lin, and Fung [443] use self-attention across different embeddings to learn multi-lingual MEs. For Named Entity Recognition (NER), the multilingual embeddings in the ensemble set are concatenated and passed as the input into a transformer encoder and a conditional random field is used as the classification layer. The use of self-attention weights showed to improve over a linear layer without self-attention weights. Winata et al. [444] have also extended this to using hierarchical MEs (HMEs), which refers to word-level MEs that are created via a weighted average of sub-word level ME representations. They find that HMEs outperform regular MEs on code-switching NER through the use of pretrained subword embeddings given by `fasttext` [187].

García, Agerri, and Rigau [120] learn MEs for cross-lingual tasks by projecting the embeddings into a common semantic space. Embeddings of resource-rich languages can then be used to improve the quality of learned embeddings of low-resourced languages. This is achieved in three steps: (1) align the vector spaces of different vocabularies of each language using the bilingual mapper `VecMap` [16], (2) create new embeddings for missing words in the source embeddings and (3) average the embeddings in the ensemble set. The resulting ME vocabulary will then be the union of the vocabularies of the word embeddings used. This method is referred to as MVM (Meta-VecMap).

Doval et al. [98] align embeddings into a bilingual vector space using `VecMap` and `MUSE` [72] and use a linear layer to transform the aligned embeddings such that the average word vector in the ensemble set can be predicted in the target language from the source. This is

motivated by the finding that the average of word embeddings is a good approximation for MEs [66] as already discussed in Equation 2.4.6.

#### 2.4.7 Reinforcement Learning Based Knowledge Distillation

Knowledge distillation has also been performed using reinforcement learning (RL) where the objective is to optimize for accumulated of rewards where the reward function can be task-specific. Since not all problems optimize for the log-likelihood, standard supervised learning can be a poor surrogate, hence RL-based distillation can directly optimize for the metric used for evaluation.

**NETWORK2NETWORK COMPRESSION** Ashok et al. [17] propose Network to Network (N2N) compression in policy gradient-based models using a RNN policy network that removes layers from the ‘teacher’ model while another RNN policy network then reduces the size of the remaining layers. The resulting policy network is trained to find a locally optimal student network and accuracy is considered the reward signal. The policy networks gradients are updated accordingly, achieving a compression ratio of 10 for ResNet-34 while maintaining similar performance to the original teacher network.

**FITNETS** Romero et al. [360] propose a student network that has deeper yet smaller hidden layers compared to the teacher network. They also constrain the hidden representations between the networks to be similar. Since the hidden layer size for student and teacher will be different, they project the student layer to into an embedding space of fixed size so that both teacher and student hidden representations are of the same size. Equation 2.60 represents the Fitnet loss where the first term represents the cross-entropy between the target  $y_{\text{true}}$  and the student probability  $P_S$ , while  $H(P_T^\gamma, P_S^\gamma)$  represents the cross entropy between the normalized and flattened teachers hidden representation  $P_T^\gamma$  and the normalized student hidden representation  $P_S^\gamma$  where  $\gamma$  controls the influence of this similarity constraint.

$$\mathcal{L}_{\text{MD}}(\mathbf{W}_S) = H(y_{\text{true}}, P_S) + \gamma H(P_T^\gamma, P_S^\gamma) \quad (2.60)$$

Equation 2.61 shows the loss between the teacher weights  $\mathbf{W}_{\text{Guided}}$  for a given layer and the reconstructed weights  $\mathbf{W}_r$  which are the weights of a corresponding student network projected using a convolutional layer (cuts down computation compared to a fully-connected projection layer) to the same hidden size of the teacher network weights.

$$\mathcal{L}_{\text{HT}}(\mathbf{W}_T, \mathbf{W}_r) = \frac{1}{2} \|u_h(\mathbf{x}; \mathbf{W}_{\text{Hint}}) - r(v_g(\mathbf{x}; \mathbf{W}_T); \mathbf{W}_r)\|^2 \quad (2.61)$$

where  $u_h$  and  $v_g$  are the teacher/student deep nested functions up to their respective hint/guided layers with parameters  $\mathbf{W}_{\text{Hint}}$  and  $\mathbf{W}_{\text{Guided}}$ ,  $r$  is the regressor function on top of the guided layer with parameters  $\mathbf{W}_r$ . Note that the outputs of  $u_h$  and  $r$  have to be comparable, i.e.,  $u_h$  and  $r$  must be the same non-linearity. The teacher tries to imitate the flow matrices from the teacher which are defined as the inner product between feature maps, such as layers in a residual block.

#### 2.4.8 Generative Modelling Based Knowledge Distillation

Here, we describe how two commonly used generative models, variational inference (VI) and generative adversarial networks (GANs), have been applied to learning a student networks.

##### 2.4.8.1 Variational Inference Learned Student

Hegde et al. [154] propose a variational student whereby VI is used for knowledge distillation. The parameters induced by using VI-based least squares objective are sparse, improving the generalizability of the student network. Sparse Variational Dropout Kingma, Salimans, and Welling [203] and Molchanov, Ashukha, and Vetrov [294] techniques can also be used in this framework to promote sparsity in the network. The VI objective is shown in Equation 2.62, where  $\mathbf{z}^s$  and  $\mathbf{z}^t$  are the output logits from student and teacher networks.

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \mathbf{y}, \mathbf{W}_s, \mathbf{W}_t, \alpha) = & -\frac{1}{N} \sum_{n=1}^N \mathbf{y}_n \log(\mathbf{z}_n^s) + \\ & \lambda_T \left[ 2T^2 D_{\text{KL}} \left( \sigma' \left( \frac{\mathbf{z}^s}{T} \right) \parallel \sigma' \left( \frac{\mathbf{z}^t}{T} \right) \right) \right] \\ & + \lambda_V \mathcal{L}_{\text{KL}}(\mathbf{W}_s, \alpha) + \lambda_g \sum_{m=1}^M \left| \max_{n,k,h,l} W_{T:S}(m, n, k, h, l) \right| \end{aligned} \quad (2.62)$$

Their training procedure and loss function used learns a compact and sparse student network. The roles of different terms in variational loss function are: likelihood - for independent student network's learning; hint - learning induced from teacher network; variational term - promotes sparsity by optimizing variational dropout parameters,  $\alpha$ ; Block Sparse Regularization - promotes and transfers sparsity from the teacher network.

##### 2.4.8.2 Generative Adversarial Student

GANs train a binary classifier  $f_w$  to discriminate between real samples  $x$  and generated samples  $g_\theta(z)$  that are given by a generator network  $g_\theta$  and  $z$  is sampled from  $p_g$  a known distribution e.g a Gaussian. A

minimax objective is used to minimize the misclassifications of the discriminator while maximizing the generators accuracy of tricking the discriminator. This is formulated as,

$$\min_{\theta \in \Theta} \max_{\mathbf{w} \in W} \mathbb{E}_{\mathbf{x} \sim p_{data}} [\log(f_{\mathbf{w}}(\mathbf{x}))] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}} [\log(1 - f_{\mathbf{w}}(g_{\theta}(\mathbf{z})))] \quad (2.63)$$

where the global minimum is found when the generator distribution  $p_g$  is similar to the data distribution  $p_{data}$  (referred to as the nash equilibrium). Wang et al. [436] learn a Generative Adversarial Student Network where the generator learns from the teacher network using the minimax objective in Equation 2.63. They reduce the variance in gradient updates which leads less epochs requires to train to convergence, by using the Gumbel-Max trick in the formulation of GAN knowledge distillation. First they propose Naive GAN (NaGAN) which consists of a classifier  $C$  and a discriminator  $D$  where  $C$  generates pseudo labels given a sample  $x$  from a categorical distribution  $p_c(\mathbf{y}|\mathbf{x})$  and  $D$  distinguishes between the true targets and the generated ones. The objective for NaGAN is expressed as,

$$\min_c \max_d V(c, d) = \mathbb{E}_{\mathbf{y} \sim p_u} [\log p_d(\mathbf{x}, \mathbf{y})] + \mathbb{E}_{\mathbf{y} \sim p_c} [\log(1 - p_d^g(\mathbf{x}, \mathbf{y}))] \quad (2.64)$$

where  $V(c, d)$  is the value function and  $C$  and  $D$  are the scoring functions of  $h(\mathbf{x}, y)$  and  $g(x, y)$  respectively. Then  $p_c(y|\mathbf{x}) = \phi(h(\mathbf{x}, y))$  and  $p_d^g(\mathbf{x}, y) = \sigma(g(\mathbf{x}, y))$  where  $\phi$  is the softmax function and  $\sigma$  is the sigmoid function. However, NaGAN requires a large number of samples and epochs to converge to nash equilibrium using this objective, since the gradients from  $D$  that update  $C$  can often vanish or explode. This brings us to their main contribution, Knowledge Distilled GAN (KDGAN). KDGAN somewhat remedy the aforementioned convergence problem by introducing a pretrained teacher network  $T$  along with  $C$  and  $D$ . The objective then consists of a distillation  $\ell_2$  loss component between  $T$  and  $C$  and adversarial loss between  $T$  and  $D$ . Therefore, both  $C$  and  $T$  aim to fool  $D$  by generating fake labels that seem real, while  $C$  tries to distil the knowledge from  $T$  such that both  $C$  and  $T$  agree on a good fake label. The student network convergence is tracked by observing the generator outputs and loss changes. Since the gradient from  $T$  tend to have low variance, this can help  $C$  converge faster, reaching a nash equilibrium.

COMPRESSING GENERATIVE ADVERSARIAL NETWORKS Aguineldo et al. [2] compress GANs achieving high compression ratios (58:1 on CIFAR-10 and 87:1 CelebA) while maintaining high Inception Score (IS) and low Frechet Inception Distance (FID). They're main finding is that a compressed GAN can outperform the original overparameterized teacher

GAN, providing further evidence for the benefit of compression in very large networks. A student learns from the teacher using a joint loss between the student GAN discriminator and teacher generator DCGAN. The teacher generator was trained using deconvolutional GAN [DCGAN; 348] framework. They use a joint training loss to optimize that can be expressed as,

$$\min_{\theta \in \Theta} \max_{\mathbf{w} \in W} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [\log(f_{\mathbf{w}}(\mathbf{x}))] + \mathbb{E}_{z \sim p_z} \left[ \alpha \log(1 - f_{\mathbf{w}}(g_{\theta}(z))) + (1 - \alpha) g_{\text{teacher}} \| (z) - g_{\theta}(z) \|^2 \right] \quad (2.65)$$

where  $\alpha$  controls the influence of the MSE loss between the logit predictions  $g_{\text{teacher}}(z)$  and  $g_{\theta}(z)$  of teacher and student respectively. The terms with expectations correspond to the standard adversarial loss.

#### 2.4.9 Pairwise-based Knowledge Distillation

Apart from pointwise classification tasks, knowledge distillation has also been performed for pairwise tasks, which we discuss in the below sections.

##### 2.4.9.1 Similarity-preserving Knowledge Distillation

Semantically similar inputs tend to have similar activation patterns. Based on this premise, Tung and Mori [419] have propose knowledge distillation such that input pair similarity scores from the student network are similar to those from the teacher network. This can be a pairwise learning extension of the standard knowledge distillation approaches.

They aim to preserve similarity between student and pretrained teacher activations for a given batch of similar and dissimilar input pairs. For a batch  $b$ , a similarity matrix  $G(l')_S \in \mathbb{R}^{b \times b}$  is produced between their student activations  $A_S^{(l')}$  at the  $l'$  layer and teacher activations  $A_T^{(l)}$  at the  $l$ -th layer. The objective is then defined as the cross entropy between the student logit output  $\sigma(\mathbf{z}_s)$  and target  $y$  summed with the similarity preserving distillation loss component on the RHS of Equation 2.66,

$$\mathcal{L} = \mathcal{L}_{ce}(\mathbf{y}, \phi(\mathbf{Z}_S)) + \frac{\gamma}{b^2} \sum_{(l, l') \in \mathcal{I}} \| \mathbf{G}_T^{(l)} - \mathbf{G}_S^{(l')} \|_F^2 \quad (2.66)$$

where  $\| \cdot \|_F$  denotes the Frobenius norm,  $\mathcal{I}$  is the total number of layer pairs considered and  $\gamma$  controls the influence of similarity preserving term between both networks.

In the transfer learning setting, their experiments show that similarity preserving can be a robust way to deal with domain shift. Moreover, this method complements the SoTA attention transfer [478] approach.

#### 2.4.9.2 Contrastive Representation Distillation

Instead of minimizing the KL divergence between the scalar outputs of teacher network  $T$  and student network  $S$ , Tian, Krishnan, and Isola [414] propose to preserve structural information of the embedding space. Similar to Hinton et al. [160], they force the representations between the student and teacher network to be similar but instead use a contrastive loss that moves positive paired representations closer together while positive-negative pairs away. This contrastive objective is given by,

$$f^{S*} = \arg \max_{f^S} \max_h \mathcal{L}_{\text{critic}}(h) = \arg \max_{f^S} \max_h \mathbb{E}_q(T, S | C = 1) [\log h(T, S)] + N \mathbb{E}_q(T, S | C = 0) [\log(1 - h(T, S))] \quad (2.67)$$

where  $h(T, S) = \frac{e^{g^T(T)'g^S(S)'/\tau}}{e^{g^T(T)'g^S(S)'/\tau} + NM}$ ,  $M$  is number of data samples,  $\tau$  is the temperature. If the dimensionality of the outputs from  $g^T$  and  $g^S$  are not equal, a linear transformation is made to fixed size followed by an  $\ell_2$  normalization. The correlations between student and teacher network are accounted for in CRD while in standard teacher-student networks ignores the correlations and to a less extent this is also found for attention transfer [480] and the student network distilled by KL divergence [162].

**DISTILLING SIMCLR** Chen et al. [58] shows that an unsupervised learned contrastive-based CNN requires 10 times less labels to for fine-tuning on ImageNet compared to only using a supervised contrastive CNN (ResNet architecture). They find a strong correlation between the size of the pretrained network and the amount of labels it requires for fine-tuning. Finally, the contrastive network is distilled into a smaller version without sacrificing little classification accuracy.

#### 2.4.9.3 Relational Knowledge Distillation

Park et al. [328] apply knowledge distillation to relational data and propose distance (huber) and angular-based (cosine proximity) loss functions that account for different relational structures and claim that metric learning allows the student relational network to outperform the teacher network on achieving SoTA on relational datasets.

The  $\psi(\cdot)$  similarity function from the relation teacher network outputs a score that is transferred to as a pseudo target for the teacher network to learn from as,

$$\delta(x, y) = \begin{cases} \frac{1}{2} \sum_{i=1}^N (x - y)^2 & \text{for } |x - y| \leq 1 \\ |x - y| - 1 & \text{otherwise} \end{cases}$$

In the case of the angular loss shown in Equation 2.68,  $e^{ij} = \frac{t_i - t_j}{\|t_i - t_j\|_2}$ ,  $e^{kj} = \frac{t_k - t_j}{\|t_k - t_j\|_2}$ .

$$\psi_A(t_i, t_j, t_k) = \cos \angle t_i t_j t_k = \langle e_{ij}, e_{kj} \rangle \quad (2.68)$$

They find that measuring the angle between teacher and student outputs as input to the huber loss  $\mathcal{L}_{\text{delta}}$  leads to improved performance when compared to previous SoTA on metric learning tasks.

$$\mathcal{L}_{\text{rmd-a}} = \sum_{(x_i, x_j, x_k) \in \mathcal{X}^3} l_\delta \psi_A(t_i, t_j, t_k), \psi_A(s_i, s_j, s_k) \quad (2.69)$$

This is then used as a regularization terms to the task specific loss as,  $\mathcal{L}_{\text{task}} + \lambda_{\text{MD}} \mathcal{L}_{\text{MD}}$ . When used in metric learning the triplet loss shown in Equation 2.70 is used.

$$\mathcal{L}_{\text{triplet}} = \left[ \|f(\mathbf{x}_a) - f(\mathbf{x}_p)\|_2^2 - \|f(\mathbf{x}_a) - f(\mathbf{x}_n)\|_2^2 + m \right]_+ \quad (2.70)$$

CORRELATION CONGRUENCE KNOWLEDGE DISTILLATION Peng et al. [333] extend pairwise KD to accounting for correlations between multiple instances using a generalized kernel method between representations of various instances. They find that using this additional KD objective that uses the kernel loss improves KD both on the output of the network or when using it on the intermediate representations, as in hint-based KD.

Passalis, Tzelepi, and Tefas [331] too use kernel methods to estimate similarity between the student and teacher networks estimated conditional probability distributions and propose various kernels such as Gaussian, Cosine and T-student.

KNOWLEDGE DISTILLATION VIA FACTOR TRANSFER Kim, Park, and Kwak [198] propose Factor Transfer (FT) to paraphrase the teachers knowledge and translate it such that student can easier learn. This is achieved through convolutional operations that learn salient factors of the teacher representations which are then learned from by the student. They find this paraphrase-translate scheme to outperform standard KD for computer vision tasks.

ATTENTION-TRANSFER KNOWLEDGE DISTILLATION Song et al. [395] use attention-based knowledge distillation for fashion matching that jointly learns to match clothing items while incorporating domain knowledge rules defined by clothing description where the attention learns to assign weights corresponding to the rule confidence.

NEURON SELECTIVITY TRANSFER KNOWLEDGE DISTILLATION Huang and Wang [176] minimize the maximum mean discrepancy (MMD) between the student and teacher intermediate feature maps. The MMD loss along with the standard cross-entropy loss is given as,

$$\begin{aligned} \mathcal{L}_{MMD2}(\mathbf{F}_T, \mathbf{F}_S) = & \frac{1}{C_T^2} C^T \sum_{i=1} C^T \sum_{i=1} k\left(\frac{f_T^i}{\|f_T^i\|}, \frac{f_T^{i'}}{\|f_T^{i'}\|}\right) + \\ & \frac{1}{C_S^2} \sum_{j=1}^{C_S} \sum_{j'=1}^{C_S} k\left(\frac{f_S^j}{\|f_S^j\|_2}, \frac{f_S^{j'}}{\|f_S^{j'}\|_2}\right) - \frac{2}{C^T C^S} \sum_{i=1}^{C^T} \sum_{j=1}^{C^S} k\left(\frac{f_T^i}{\|f_T^i\|_2}, \frac{f_S^j}{\|f_S^j\|_2}\right) \end{aligned} \quad (2.71)$$

VARIATIONAL INFORMATION KNOWLEDGE DISTILLATION Ahn et al. [4] transfer knowledge by maximizing mutual information between student and teacher representations approximated through the use of a variational lower-bound objective. They find improvements over KD baselines for experiments on CIFAR-10, setting a SoTA at the time of publication.

## 2.5 QUANTIZATION

Quantization is the process of representing values with a reduced number of bits. In neural networks, this corresponds to weights, activations and gradient values. Typically, when training on the GPU, values are stored in 32-bit floating point (FP) single precision. *Half-precision* for floating point (FP-16) and integer arithmetic (INT-16) are also commonly considered. INT-16 provides higher precision but a lower dynamic range compared to FP-16. In FP-16, the result of a multiplication is accumulated into a FP-32 followed by a down-conversion to return to FP-16.

To speed up training, faster inference and reduce bandwidth memory requirements, ongoing research has focused on training and performing inference with lower-precision networks using integer precision (IP) as low as INT-8 INT-4, INT-2 or 1 bit representations [81]. Designing such networks makes it easier to train such networks on CPUs, FPGAs, ASICs and GPUs.

Two important features of quantization are the range of values that can be represented and the bit spacing. For the range of signed integers with  $n$  bits, the range is  $[-2n - 1, 2n - 2]$  and for full precision (FP-32) the range is  $[-3.4 \times 1038, +3.4 \times 1038]$ . For signed integers, there  $2n$

values in that range and approximately  $4.2 \times 10^9$  for FP-32. FP can represent a large array of distributions which is useful for neural network computation, however this comes at larger computational costs when compared to integer values. For integers to be used to represent weight matrices and activations, a FP scale factor is often used, hence many quantization approaches involve a hybrid of mostly integer formats with FP-32 scaling numbers. This approach is often referred to as mixed-precision (MP) and different MP strategies have been used to avoid overflows during training and/or inference of low resolution networks given the limited range of integer formats.

In practice, this often requires the storage of hidden layer outputs with full-precision (or at least with represented with more bits than the lower resolution copies). The main forward-pass and backpropagation is carried out with lower resolution copies and convert back to the full-precision stored “accumulators” for the gradient updates.

In the extreme case where binary weights (-1, 1) or 2-bit ternary weights (-1, 0, 1) are used in fully-connected or convolutional layers, multiplications are not used, only additions and subtractions. For binary activations, bitwise operations are used [353] and therefore addition is not used. For example, Rastegari et al. [353] proposed XNOR-Networks, where binary operations are used in a network made up of xnor gates which approximate convolutions leading to 58 times speedup and 32 times memory savings.

### 2.5.1 *Approximating High Resolution Computation*

Quantizing from FP-32 to 8-bit integers with retraining can result in an unacceptable drop in performance. Retraining quantized networks has shown to be effective for maintaining accuracy in some works [136]. Other work [88] compress gradients and activations from FP-32 to 8 bit approximations to maximize bandwidth use and find that performance is maintained on MNIST, CIFAR10 and ImageNet when parallelizing both model and data.

The quantization ranges can be found using k-means quantization [259], product quantization [185] and residual quantization [50]. Fixed point quantization with optimized bit width can reduce existing networks significantly without reducing performance and even improve over the original network with retraining [240].

Courbariaux, Bengio, and David [74] instead scale using shifts, eliminating the necessity of floating point operations for scaling. This involves an integer or fixed point multiplication, as apart of a dot product, followed by the shift.

Dettmers [88] have also used FP-32 scaling factors for INT-8 weights and where the scaling factor is adapted during training along with the activation output range. They also consider not adapting the min-max ranges online and clip outlying values that may occur as a result

of this in order to drastically reduce the min-max range. They find SoTA speedups for CNN parallelism, achieving a 50 time speedup over baselines on 96 GPUs.

Gupta et al. [135] show that stochastic rounding techniques are important for FP-16 DNNs to converge and maintain test accuracy compared to their FP-32 counterpart models. In stochastic rounding the weight  $x$  is rounded to the nearest target fixed point representation  $[x]$  with probability  $1 - (x - [x])/\epsilon$  where  $\epsilon$  is the smallest positive number representable in the fixed-point format, otherwise  $x$  is rounded to  $x + \epsilon$ . Hence, if  $x$  is close to  $[x]$  then the probability is higher of being assigned  $[x]$ . Wang et al. [435] train DNNs with FP-8 while using FP-16 chunk-based accumulations with the aforementioned stochastic rounding hardware.

The necessity of stochastic rounding, and other requirements such as loss scaling, has been avoided using customized formats such as Brain float point [(BFP) 190] which use FP-16 with the same number of exponent bits as FP-32. Cambier et al. [51] recently propose a shifted and squeezed 8-bit FP format (S2FP-8) to also avoid the need of stochastic rounding and loss scaling, while providing dynamic ranges for gradients, weights and activations. Unlike other related 8-bit techniques [276], the first and last layer do not need to be in FP32 format, although the accumulator converts the outputs to FP32.

### 2.5.2 Adaptive Ranges and Clipping

Park, Yoo, and Vajda [326] exploit the fact that most the weight and activation values are scattered around a narrower region while larger values outside such region can be represented with higher precision.

They propose 3-bit activations for training quantized ResNet and Inception CNN architectures during retraining. For inference on this retrained low precision trained network, weights are also quantized to 4-bits for inference with 1% of the network being 16-bit scaling factor scalars, achieving accuracy within 1% of the original network. This was also shown to be effective in LSTM network on language modelling, achieving similar perplexities for bitwidths of 2, 3 and 4.

Migacz [280] use relative entropy to measure the loss of information between two encodings and aim minimize the KL divergence between activation output values. For each layer they store histograms of activations, generate quantized distributions with different saturation thresholds and choose the threshold that minimizes the KL divergence between the original distribution and the quantized distribution.

Banner et al. [21] analyze the tradeoff between quantization noise and clipping distortion and derive an expression for the mean-squared error degradation due to clipping. Optimizing for this results in choosing clipping values that improve 40% accuracy over standard quantization of VGG16-BN to 4-bit integer.

Another approach is to use scaling factors per group of weights (e.g channels in the case of CNNs or internal gates in LSTMs) as opposed to whole layers, particularly useful when the variance in weight distribution between the weight groupings is relatively high.

### 2.5.3 Robustness to Quantization and Related Distortions

Merolla et al. [278] have studied the effects of different distortions on the weights and activations, including quantization, multiplicative noise (akin to Gaussian DropConnect), binarization (sign) along with other nonlinear projections and simply clipping the weights. This suggests that neural networks are robust to such distortions at the expense of longer convergence times.

In the best case of these distortions, they can achieve 11% test error on CIFAR-10 with 0.68 effective bits per weight. They find that training with weight projections other than quantization performs relatively well on ImageNet and CIFAR-10, particularly their proposed stochastic projection rule that leads to 7.64% error on CIFAR-10.

Others have also shown DNNs robustness to training binary and ternary networks [74, 135], albeit a larger number of bit weight and ternary weights that are required.

### 2.5.4 Retraining Quantized Networks

Thus far, these post-training quantization (PTQ) methods without retraining are mostly effective on overparameterized models. For smaller models that are already restricted by the degrees of freedom, PTQ can lead to relatively large performance degradation in comparison to the overparameterized regime, which has been reflected in recent findings that architectures such as MobileNet suffer when using PTQ to 8-bit integer formats and lower [181, 208].

Hence, retraining is particularly important as the number of bits used for representation decreases e.g 4-bits with range  $[-8, 8]$ . However, quantization results in discontinuities which makes differentiation during backpropagation difficult. To overcome this limitation, Zhou et al. [492] quantized gradients to 6-bit number and stochastically propagate back through CNN architectures such as AlexNet using straight through estimators, defined as Equation 2.72. Here, a real number input  $r_i \in [0, 1]$  to a  $n$ -bit number output  $r_o \in [0, 1]$  and  $\mathcal{L}$  is the objective function.

$$\text{Forward} : r_o = \frac{1}{2^n - 1} \text{round}((2^n - 1)r_i) \quad (2.72)$$

$$\text{Backward} : \frac{\partial \mathcal{L}}{\partial r_i} = \frac{\partial \mathcal{L}}{\partial r_o} \quad (2.73)$$

To compute the integer dot product of  $r_0$  with another  $n$ -bit vector, they use Equation 2.74, with a computational complexity of  $\mathcal{O}(MK)$ ,

directly proportional to bitwidth of  $x$  and  $y$ . Furthermore, bitwise kernels can also be used for faster training and inference

$$x \cdot y = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} 2^{m+k} \text{bitcount}[\text{and}(c_m(\mathbf{x}), c_k(\mathbf{y}))] \quad (2.74)$$

$$c_m(\mathbf{x})_i, c_k(\mathbf{y}) \quad i \in \{0, 1\} \quad \forall i, m, k \quad (2.75)$$

**MODEL DISTILLED QUANTIZATION** Polino, Pascanu, and Alistarh [346] use a distillation loss with respect to the teacher network whose weights are quantized to set number of levels and quantized teacher trains the ‘student’. They also propose differentiable quantization, which optimizes the location of quantization points through SGD, to better fit the behavior of the teacher model.

**QUANTIZING UNBOUNDED ACTIVATION FUNCTIONS** When the nonlinear activation unit used is not bounded in a given range, it is difficult to choose the bit range. Unlike sigmoid and tanh functions that are bounded in  $[0, 1]$  and  $[-1, 1]$  respectively, the ReLU function is unbounded in  $[0, \infty]$ . Obviously, simply avoiding such unbounded functions is one option, another is to clip values outside an upper bound [288, 492] or dynamically update the clipping threshold for each layer and set the scaling factor for quantization accordingly [65].

**MIXED PRECISION TRAINING** Mixed Precision Training (MPT) is often used to train quantized networks, whereby some values remain in full-precision so that performance is maintained and some of the aforementioned problems (e.g overflows) do not cause divergent training. It has also been observed that activations are more sensitive to quantization than weights [492]

Micikevicius et al. [279] use half-precision (16-bit) floating point accuracy to represent weights, activations and gradients, without losing model accuracy or having to modify hyperparameters, almost halving the memory requirements. They round a single-precision copy of the weights for forward and backward passes after performing gradient-updates, use loss-scaling to preserve small magnitude gradient values and perform half-precision computation that accumulates into single-precision outputs before storing again as half-precision in memory.

The forward and backward passes are performed with FP-16 precision copies. Once the backward pass is performed the computed FP-16 gradients are used to update the original FP-32 precision master weight. After training, the quantized weights are used for inference along with quantized activation units. This can be used in any type of layer, convolutional or fully-connected.

Others have focused solely on quantizing weights, keeping the activations at FP32 [233, 493]. During gradient descent, Zhu et al. [493] learn

both the quantized ternary weights and pick which of these values is assigned to each weight, represented in a codebook.

Das et al. [82] propose using Integer Fused-Multiply-and-Accumulate (FMA) operations to accumulate results of multiplied INT-16 values into INT-32 outputs and use dynamic fixed point scheme to use in tensor operations. This involves the use of a shared tensor-wide exponent and down-conversion on the maximum value of an output tensor at each given training iteration using stochastic, nearest and biased rounding. They also deal with overflow by proposing a scheme that accumulates INT-32 intermediate results to FP-32 and can trade off between precision and length of the accumulate chain to improve accuracy on the image classification tasks. They argue that previous reported results on mixed-precision integer training report on non-SoTA architectures and less difficult image tasks and hence they also report their technique on SoTA architectures for the ImageNet 1K dataset.

**QUANTIZING BY ADAPTING THE NETWORK STRUCTURE** To further improve over mixed-precision training, there has been recent work that have aimed at better simulating the effects of quantization during training.

Mishra and Marr [287] combine low bit precision and knowledge distillation using three different schemes: (1) a low-precision (4-bit) ResNet network is trained from a full-precision ResNet network both from scratch, (2) a full precision trained network is transferred to train a low-precision network from scratch and (3) a trained full-precision network guides a smaller full-precision student randomly initialized network which is gradually becomes lower precision throughout training. They find that (2) converges faster when supervised by an already trained network and (3) outperforms (1) and set at that time was SoTA for Resnet classifiers at ternary and 4-bit precision.

Lin, Zhao, and Pan [245] replace FP-32 convolutions with multiple binary convolutions with various scaling factors for each convolution, overall resulting in a large range.

Zhou et al. [492] and Choi et al. [65] have both reported that the first and last convolutional layers are most sensitive to quantization and hence many works have avoided quantization on such layers. However, Choi et al. [65] find that if the quantization is not very low (e.g 8-bit integers) then these layers are expressive enough to maintain accuracy.

Zhou et al. [488] have overcome this problem by iteratively quantizing the network instead of quantize the whole model at once. During the retraining of an FP-32 model, each layer is iteratively quantized over consecutive epochs. They also consider using supervision from a teacher network to learn a smaller quantized student network, combining knowledge distillation with quantization for further reductions.

QUANTIZATION WITH PRUNING & HUFFMAN CODING Coding schemes can be used to encode information in an efficient manner and construct codebooks that represent weight values and activation bit spacing. Han, Mao, and Dally [144] use pruning with quantization and huffman encoding for compression of ANNs by 35-49 times (9-13 times for pruning, quantization represents the weights in 5 bits instead of 32) the original size without affecting accuracy.

Once the pruned network is established, the parameter are quantized to promote weight sharing. This multi-stage compression strategy combines weight sharing and fine-tuning of centroids. They note that too much pruning on channel level sparsity (as opposed to kernel-level) can effect the network’s representational capacity.

#### 2.5.4.1 Loss-aware quantization

[166] propose a proximal Newton algorithm with a diagonal Hessian approximation to minimize the loss with respect to the binarized weights  $\hat{\mathbf{w}} = \alpha \mathbf{b}$ , where  $\alpha > 0$  and  $\mathbf{b}$  is binary. During training,  $\alpha$  is computed for the  $l$ -th layer at the  $t$ -th iteration as  $\alpha_l^t = \|\mathbf{d}^{t-1} \otimes \mathbf{w}_l^t\|_1 / \|\mathbf{d}_l^{t-1}\|_1$  where  $\mathbf{d}_l^{t-1} := \text{diag}(\mathbf{D}_l^{t-1})$  and  $\mathbf{b}_l^t = (\mathbf{w}_l^t)$ . The input is then rescaled for layer  $l$  as  $\tilde{\mathbf{x}}_l^t = \alpha_l^t \mathbf{x}_{l-1}^t$  and then compute  $\mathbf{z}_l^t$  with input  $\tilde{\mathbf{x}}_{l-1}^t$  and binary weight  $\mathbf{b}_l^t$ .

Equation 2.76 shows the proximal newton update step where  $w_l^t$  is the weight update at iteration  $t$  for layer  $l$ ,  $\mathbf{D}$  is an approximation to the diagonal of the Hessian which is already given as the  $2^{nd}$  momentum of the adaptive momentum (adam) optimizer. The  $t$ -th iteration of the proximal Newton update is as follows:

$$\begin{aligned} \min_{\hat{\mathbf{w}}^t} \nabla \ell(\hat{\mathbf{w}}^{t-1})^T (\hat{\mathbf{w}}^t - \hat{\mathbf{w}}^{t-1}) + (\hat{\mathbf{w}}^t - \hat{\mathbf{w}}^{t-1}) \mathbf{D}^{t-1} (\hat{\mathbf{w}}^t - \hat{\mathbf{w}}^{t-1}) \\ \text{s.t. } \hat{\mathbf{w}}_l^t = \alpha_l^t \mathbf{b}_l^t, \alpha_l^t > 0, \mathbf{b}_l^t \in \{+/-1\}^{n_l}, l = 1, \dots, L. \end{aligned} \quad (2.76)$$

where the loss  $\ell$  w.r.t binarized version of  $\ell(w_t)$  is expressed in terms of the  $2^{nd}$ -order TS expansion using a diagonal approximation of the Hessian  $\mathbf{H}^{t-1}$ , which estimates of the Hessian at  $\mathbf{w}^{t-1}$ . Similar to the  $2^{nd}$  order approximations discussed in Section 2.2.2, the Hessian is essential since  $\ell$  is often flat in some directions but highly curved in others.

EXPLICIT LOSS-AWARE QUANTIZATION Zhou et al. [489] propose an Explicit Loss-Aware Quantization (ELQ) method that minimizes the loss perturbation from quantization in an incremental way for very low bit precision i.e binary and ternary. Since going from FP-32 to binary or ternary bit representations can cause considerable fluctuations in weight magnitudes and in turn the predictions, ELQ directly incorporates this quantization effect in the loss function as

$$\min_{\hat{\mathbf{W}}_l} +a_1 \mathcal{L}_p(\mathbf{W}_l, \hat{\mathbf{W}}_l) + E(\mathbf{W}_l, \hat{\mathbf{W}}_l) \quad (2.77)$$

where  $\hat{W} \in \{a_l c_k | 1 \leq k \leq K\}$ ,  $1 \leq l \leq L$ ,  $L_p$  is the loss difference between quantized and the original model  $\|\mathcal{L}(\mathbf{W}_l) - \mathcal{L}(\hat{\mathbf{W}}_l)\|$ ,  $E$  is the reconstruction error between the quantized and original weights  $\|\mathbf{W}_l - \hat{\mathbf{W}}_l\|^2$ ,  $a_l$  a regularization coefficient for the  $l$ -th layer and  $c_k$  is an integer and  $k$  is the number of weight centroids.

VALUE-AWARE QUANTIZATION Park, Yoo, and Vajda [326] like prior work mentioned in this work have also succeeded in reduced precision by reducing the dynamic range by narrowing the range where most of the weight values concentrate. Different to other work, they assign higher precision to the outliers as opposed to mapping them to the extremum of the reduced range. This small difference allow 3-bit activations to be used in ResNet-152 and DenseNet-201, leading to a 41.6% and 53.7% reduction in network size respectively.

#### 2.5.4.2 Differentiable Quantization

When considering fully-differentiable training with quantized weight and activations, it is not obvious how to back-propagate through the quantization functions. These functions are discrete-valued, hence their derivative is 0 almost everywhere. So, using their gradients as-is would severely hinder the learning process. A commonly used approximation to overcome this issue is the “straight-through estimator” (STE) [32, 161], which simply passes the gradient through these functions as-is, however there has been a plethora of other techniques proposed in recent years which we describe below.

DIFFERENTIABLE SOFT QUANTIZATION Gong et al. [126] have proposed differentiable soft quantization (DSQ) learn clipping ranges in the forward pass and approximating gradients in the backward pass. To approximate the derivative of a binary quantization function, they propose a differentiable asymptotic function (i.e smooth) which is closer to the quantization function than it is to a full-precision  $\tanh$  function and therefore will result in less of a degradation in accuracy when converted to the binary quantization function post-training.

For multi-bit uniform quantization, given the bit width  $b$  and the floating-point activation/weight  $\mathbf{x}$  following in the range  $(l, u)$ , the complete quantization-dequantization process of uniform quantization can be defined as:  $Q_U(\mathbf{x}) = \text{round}(\mathbf{x}\Delta)\Delta$  where the original range  $(l, u)$  is divided into  $2^b - 1$  intervals  $\mathcal{P}_i, i \in (0, 1, \dots, 2^b - 1)$ , and  $\Delta = \frac{u-l}{2^b-1}$  is the interval length. The DSQ function, shown in Equation 2.78, handles the point  $\mathbf{x}$  depending what interval in  $\mathcal{P}_i$  lies.

$$\phi(x) = s \tanh(k(\mathbf{x} - m_i)), \quad \text{if } \mathbf{x} \in \mathcal{P}_i \quad (2.78)$$

with

$$m_i = l + (i + 0.5)\Delta \quad \text{and} \quad s = 1 \tanh(0.5k\Delta) \quad (2.79)$$

The scale parameter  $s$  for the tanh function  $\varphi$  ensures a smooth transitions between adjacent bit values, while  $k$  defines the functions shape where large  $k$  corresponds close to consecutive step functions given by uniform quantization with multiple piecewise levels. The DSQ function then approximates the uniform quantizer  $\varphi$  as follows:

$$\mathcal{Q}_S(\mathbf{x}) = \begin{cases} l, & \mathbf{x} < l, \\ u, & \mathbf{x} > u, \\ l + \Delta(i + \frac{\varphi(\mathbf{x})+1}{2}), & \mathbf{x} \in \mathcal{P}_i \end{cases}$$

The DSQ can be viewed as aligning the data with the quantization values with minimal quantization error due to the bit spacing that is carried out to reflect the weight and activation distributions. Standard quantization is near perfectly approximated when the largest value on the curve bounded by  $+1$  is small. They introduce a characteristic variable  $\alpha := 1 - \tanh(0.5k\Delta) = 1 - \frac{1}{s}$  and given that,

$$\varphi(0.5\Delta) = 1 \quad \Rightarrow \quad k = \frac{1}{\Delta} \log(2/\alpha - 1), \quad \Delta = \frac{u-l}{2^b - 1} \quad (2.80)$$

DSQ can be used as a piecewise uniform quantizer and when only one interval is used, it is the equivalent of using DSQ for binarization.

**SOFT-TO-HARD VECTOR QUANTIZATION** Agustsson et al. [3] propose to compress both the feature representations and the model by gradually transitioning from soft to hard quantization during retraining and is end-to-end differentiable. They jointly learn the quantization levels with the weights and show that vector quantization can be improved over scalar quantization.

$$H(E(\mathbf{Z})) = -X_{e \in [L]} m P(E(\mathbf{Z}) = e) \log(P(E(\mathbf{Z}) = e)) \quad (2.81)$$

They optimize the rate distortion trade-off between the expected loss and the entropy of  $\mathbf{E}(\mathbf{Z})$ :

$$\min_{E, D, \mathbf{W}} \mathbb{E}_{X, Y} [\ell(\hat{F}(X), Y) + \lambda R(\mathbf{W})] + \beta H(E(\mathbf{Z})) \quad (2.82)$$

**ITERATIVE PRODUCT QUANTIZATION (IPQ)** Quantizing a whole network at once can be too severe for low precision ( $< 8$  bits) and can lead to *quantization drift* - when scalar or vector quantization leads to an accumulation of reconstruction errors within the network that compound and lead to large performance degradations. To combat this, Stock et al. [398] iteratively quantize the network starting with low layers and only performing gradient updates on the rest of the

remaining layers until they are robust to the quantized layers. This is repeated until quantization is carried out on the last layer, resulting in the whole network being amenable to quantization. The codebook is updated by averaging the gradients of the weights within the block  $b_{\text{KL}}$  as

$$\mathbf{c} \leftarrow \mathbf{c} - \eta \frac{1}{|J_{\mathbf{c}}|} \sum_{(k,l) \in J_{\mathbf{c}}} \frac{\partial \mathcal{L}}{\partial b_{\text{KL}}} \quad \text{where} \quad J_{\mathbf{c}} = \{(k,l) \mid c[\mathbf{I}_{\text{KL}}] = \mathbf{c}\} \quad (2.83)$$

where  $\mathcal{L}$  is the loss function,  $\mathbf{I}_{\text{KL}}$  is an index for the  $(k,l)$  subvector and  $\eta > 0$  is the codebook learning rate. This adapts the upper layers to the drift appearing in their inputs, reducing the impact of the quantization approximation on the overall performance.

**QUANTIZATION-AWARE TRAINING** Instead of iPQ, Jacob et al. [181] use a straight through estimator [(STE) 32] to backpropagate through quantized (8-bit) weights and activations of convolutional layers during training with a 32-bit integer accumulator.

They also note that in order to have a challenging architecture to compress, experiments should move towards trying to compress architectures which are already have a minimal number of parameter and perform relatively well to much larger predecesing architectures e.g EfficientNet, SqueezeNet and ShuffleNet.

**QUANTIZATION NOISE** Fan et al. [105] argue that both iPQ and QAT are less suitable for very low precision such as INT4, ternary and binary. They instead propose to randomly simulate quantization noise on a subset of the network and only perform backward passes on the remaining weights in the network. Essentially this is a combination of DropConnect (instead of the Bernoulli function, it is a quantization noise function) and Straight Through Estimation is used to backpropagate through the sample of subvectors chosen for quantization for a given mini-batch.

Estimating quantization noise through randomly sampling blocks of weights to be quantized allows the model to become robust to very low precision quantization without being too severe, as is the case with previous quantization-aware training [181]. The authors show that this iterative quantization approach allows large compression rates in comparison to QAT while staying close to (few perplexity points in the case of language modelling and accuracy for image classification) the uncompressed model in terms of performance. They reach SoTA compression and accuracy tradeoffs for language modelling (compression of Transformers such as RoBERTa on WikiText) and image classification (compressing EfficientNet-B3 by 80% on ImageNet).

**HESSIAN-BASED QUANTIZATION** The precision and order (by layer) of quantization has been chosen using  $2^{\text{nd}}$  order information

from the Hessian [97]. They show that on already relatively small CNNs (ResNet20, Inception-V3, SqueezeNext) that Hessian Aware Quantization (HAWQ) training leads to SoTA compression on CIFAR-10 and ImageNet with a compression ratio of 8 and in some cases exceed the accuracy of the original network with no quantization.

Similarly, Shen et al. [384] quantize transformer-based models such as BERT with mixed precision by also using  $2^{nd}$  order information from the Hessian matrix. They show that each layer exhibits varying amount of information and use a sensitivity measure based on mean and variance of the top eigenvalues. They show the loss landscape as the two most dominant eigenvectors of the Hessian are perturbed and suggest that layers that shower a smoother curvature can undergo lower but precision. In the cases of MNLI and CoNLL datasets, upper layers closer to the output show flatter curvature in comparison to lower layers. From this observation, they are motivated to perform a group-wise quantization scheme whereby blocks of a matrix have different amounts of quantization with unique quantization ranges and look up table. A Hessian-based mixed precision scheme is then used to decide which blocks of each matrix are assigned the corresponding low-bit precisions of varying ranges and analyse the differences found for quantizing different parts of the self-attention block (self-attention matrices and fully-connected feedforward layers) and their inputs (embeddings) and find the highest compression ratios can be attributed to most of the parametes in the self-attention blocks.

## 2.6 SUMMARY

In summary, this chapter has thoroughly gone through the background to various compression techniques, including pruning, tensor decomposition, knowledge distillation, quantization and works that aim to combine the aforementioned methods. Pruning is a straightforward and useful method to reduce the size of the network, but unstructured pruning suffers from not having specialized hardware to take full advantage of sparse neural networks. Tensor decomposition can be an attractive alternative but the computation associated with decomposing high-dimensional tensors may outweigh the compression benefits given that tensor decomposition requires many retraining step to recover performance. Knowledge distillation is an alternative method that introduces a new student network that learns from an already trained network and has the benefit of not sparsifying the network, but requires the introduction of more student parameters to perform distillation training. Quantization does not introduce more parameters or sparsify the network can be easily applied in theory but current software does not interface well GPUs and TPUs for using integer precision and lower precision that is required for faster inference.



# 3

---

## MITIGATING EXPOSURE BIAS

---

This chapter focuses on addressing compounding errors in sequence predictors, a problem known as *exposure bias*. This occurs when the training time behaviour does not match the test time behavior as the ground truth is used at each timestep during training but the model must rely on past predictions as input at test time. This discrepancy results in the model accumulating errors along a predicted sequence at test time.

Nearest-neighbor replacement sampling is proposed to mitigate exposure bias by exposing the model to semantic neighbors of the ground truth along with predictions at training time to bridge the behavioural difference at test time. Below begins with an introduction, before describing the methodology and results.

### 3.1 INTRODUCTION

Many tasks in natural language require prediction in large and loosely structured output spaces. This include LM-based problems such as Neural Machine Translation (NMT), speech recognition, image captioning and question answering [282, 404] and more structured problems such as Dependency Parsing (DP), Named Entity Recognition (NER). The standard procedure for learning Recurrent Neural Network (RNN) sequence predictors for these tasks is to use Maximum Likelihood Estimation (MLE), or more specifically the conditional log-likelihood. It involves predicting the next token given past tokens in a sequence using a parametric model,  $f_{\theta}(\cdot)$ , parameterized by  $\theta$ .

This is distinctly different from the standard supervised learning that assumes the input distribution to be i.i.d., which leads to a non-convex objective even when the loss is convex given the inputs. Thus, this problem leads to a compounding of errors along a generated sequence and breaks the i.i.d assumption in supervised learning. It is often referred to as exposure bias, as the model observes the ground truth during training and biases learning towards always observing the ground truth.

In the worst case, this can lead to errors that are quadratic in the sequence length. This is intuitive since early errors may lead to transitioning to states that have yet to be observed in the training set, which is severe for high dimensional output spaces. Another issue is that

MLE treats incorrect predictions as equally wrong despite the semantic distance from the target tokens. This 0-1 type loss is problematic for high-dimensional output spaces (e.g MT and LM) as classes become more similar and the decision boundaries more difficult to separate. MLE also ignores the structure in the output space since, in the context of natural language, the loss is on the token level and ignores any interactions among the outputs.

One could argue that in cases where the learner finds it difficult to learn from the teacher policy alone, **sampling alternative inputs that are similar to the original input under a suitable curriculum can lead to improved out-of-sample performance, and can be considered as an exploration strategy**. Moreover, interpolating this scheme with past predictions further improves performance. Past findings use outputs other than that provided by an expert can be beneficial for generation tasks [162, 314] and hence, a motivation for this work.

A natural question to ask at this point is whether can be further mitigated using other sampling techniques that bridge the gap between training and test time behaviour.

Hence, to improve generalization in neural language models and address compounding errors, **this chapter describes our proposed Nearest Neighbor Replacement Sampling (NNRS) – a curriculum learning-based method that gradually changes an initially deterministic teacher policy to a stochastic policy**. A token at a given time-step is replaced with a sampled nearest neighbor of the past target with a truncated probability proportional to the cosine similarity between the original word and its top- $k$  most similar words. This allows the learner to explore alternatives when the current policy provided by the teacher is sub-optimal or difficult to learn from. The proposed method is straightforward, online and requires little additional memory requirements. The findings are reported on two language modelling benchmarks and find that the proposed method further improves performance when used in conjunction with SS [31].

### 3.2 RELATED WORK

The most relevant prior work to ours is SS, which also uses an on-line sampling strategy to reduce compounding errors. To alleviate this problem, SS alternates between  $\hat{y}_{t-1}$  and  $y_{t-1}$  using a sampling schedule, whereby the probability of using  $\hat{y}_{t-1}$  instead of  $y_{t-1}$  increases throughout training, allowing the learner to generate multi-step ahead predictions by improving the models’ robustness with respect to its own prediction errors.

Past work [362] has shown that this bound on the error can be alleviated by iterating over the dataset where past predictions are appended to the original dataset and interpolated, which results in an

error bound that is linear for some problems. However, this theoretical guarantee is relative to the optimal policy with respect to the expert policy. In NLP tasks such as machine translation and language modeling, the expert policy itself may not be enough for the learner to learn from given the high dimensionality and sparseness of the policy (i.e output) space  $\Pi$ , not to mention that this bound does not hold for the class of problems of interest (sequence prediction with high dimensional and sparse outputs). This can also be due to the gap between the learner and the expert is too large to bridge and therefore never reaches  $\pi^*$  given the non-convexity of the loss function and in the case of task-specific scores, ML is only a surrogate loss for the actual measure used. Moreover, incorrect predictions are considered equally poor which is inefficient, particularly when dealing with modelling a dirac distribution (predicting a one hot vector where all mass is given to a single token).

*Dataset Aggregation* [DAgger; 363] finds a stationary deterministic policy by allowing the model to first make predictions at test time and then queries the teacher as to what actions it could have taken given the observed errors the model made on the validation data. DAgger attempts to address compounding errors by initially using an expert policy to generate a set of sequences. These samples are added to the dataset  $\mathcal{D}$  and a new policy is learned, which subsequently generates more samples appended to  $\mathcal{D}$ . This process is repeated until convergence and the resultant policy  $\pi$  that best emulates the expert policy is selected. In the initial stages of learning, a modified policy is considered  $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i$ , where the expert  $\pi^*$  is used for a portion of the time (referred to as a mixture expert policy) so that the generated trajectories in the initial stages of training by  $\hat{\pi}_i$  at time  $i$  do not diverge to irrelevant states.

Likewise, *Mixed Incremental Cross-Entropy Reinforce* [MIXER; 352] applies REINFORCE for text generation. MIXER also uses a mixed expert policy (originally inspired by DAgger), whereby incremental learning is used with REINFORCE and cross-entropy (CE). The main difference between MIXER and SS is that the former relies on past target tokens in prediction and the latter uses REINFORCE to determine if the predictions lead to a sufficiently good return from a task-specific score. In the proposed method, it instead dynamically interpolates between the model’s past predictions, past targets and the additional intermediate step of using past target neighbors and use individual schedules for the past prediction and the past targets’  $k$ -neighbors.

Recent work also suggests that more complex architectures that were previously thought to significantly outperform standard Long Short-Term Memory (LSTM) networks (the most commonly used recurrent neural network used in natural language processing tasks), do not when LSTM hyperparameters are optimally tuned [275] using a batched GP bandits using the expected improvement acquisition function [87]. This further motivates us to move towards focusing more on optimization

and objective functions for better generalization in sequence prediction, as opposed to increasing parameter count, which does not directly address the problem with MLE training in sequence prediction.

### 3.3 METHODOLOGY

In self-supervised and supervised learning, the standard modeling procedure<sup>1</sup> involves training a policy  $\hat{\pi}$  to perform actions given an *expert* policy  $\pi^*$  at each time step  $t \in T$ . This approach is also known as *teacher forcing* [442]. However, using the past targets  $y_{t-n}$  and current input  $x_t$  to predict  $y_{t+1}$  at training time does not emulate the process at test time where the model is instead required to use its own past predictions  $\hat{y}_{t-n}$  to generate the sequence. To address this, we augment the teacher policy by sampling neighboring word vectors within a chosen radius with probability assigned via the normalized cosine similarities between each of the  $k$  neighbors and a corresponding input  $x_t$ . By using pretrained word embeddings as the basis for choosing the neighborhood for each word, we only require to store an embedding matrix  $\mathbf{E} \in \mathbb{R}^{|V| \times d}$  where  $|V|$  is the vocabulary size and the  $d$  is the dimensionality of the embedding space. In prediction-based word embeddings, typically  $d \leq |V|$ , which results in a significantly smaller memory footprint when using  $\mathbf{E}$  than using a transition probability matrix computed from a co-occurrence matrix in  $\mathbb{N}^{|V| \times |V|}$ . During training we monotonically increase the replacement probability using a number of simple non-parametric functions. We show that performance can be further improved when NNRS is used in conjunction with SS [31], a standard technique that also addresses the compounding error problem.

We consider sequence modelling tasks where the training set contains input-output pairs  $(X, Y)$ , where  $X = x_1, x_2, \dots, x_T$  is an input sequence of length  $T$  and  $Y = y_1, y_2, \dots, y_T$  is its corresponding output sequence of the same length. We use the compact notation  $x_{1:T}$  to denote the sequence  $X = x_1, x_2, \dots, x_T$ . In language modelling (LM), the task is to predict the next token in the sequence,  $x_t$ , given  $x_{1:t-1}$ . For this reason, LM can be seen as a special (unsupervised) instance of the structure prediction problem where  $Y = x_{2:T}$  and the input sequence is defined as  $X = x_{1:T-1}$ . Different models have been proposed in the literature for sequence-to-sequence prediction problem such as variants of recurrent neural networks [RNNs; 403] and Transformers [424]. We model these methods collectively as first computing a representation  $\mathbf{h}_t$  for the sequence  $x_{1:t-1}$  using some (recurrent in the case of RNNs and attention-weighted sum in the case of Transformers) function,  $f(x_{1:t-1}; \boldsymbol{\theta})$ , parameterised by  $\boldsymbol{\theta}$ . Next, the prediction,  $\hat{y}_t$ , of the next target output,  $y_t$ , is modelled as a classification problem where a probability distribution,  $p(\cdot)$ , over the possible output labels is com-

<sup>1</sup> In the context of RL we will refer to tokens  $x$  as actions  $a$ , models  $f$  as policies  $\pi$  and predictions  $\hat{y}$  as  $\pi(s'|a, s)$ .

puted (e.g., using `softmax` function) and the label with the maximum probability is selected as  $\hat{y}_t$ . Then, the parameters  $\theta$  of the generator can be learned such that the log-likelihood of the output sequence  $Y$  given by (3.1) is maximized.

$$\frac{1}{T} \sum_{t=1}^T \log p(y_t | y_{1:t-1}, X; \theta) \quad (3.1)$$

As a concrete example of this setting, let us consider NLM using an RNN. The recurrent function in the neural network updates its hidden state vector at each time step. Language modelling can be seen as a classification problem where each word in the output vocabulary,  $V$ , is a candidate label, and we must select the most probable output label (word) as the next prediction. Given the hidden state vector  $\mathbf{h}_{t-1}$ , after observing the sequence  $x_{1:t-1}$ , we can compute the probability of each word  $w \in V$  using  $\sigma(\mathbf{h}_t^\top \theta_w)$ , where  $\theta_w$  is a parameter vector (embedding) corresponding to  $w$  and  $\sigma$  is the `softmax` function.

Finally, the log-likelihood of the given corpus is computed using (3.1) and stochastic gradient descent (SGD) (or a variant) can be used to find the optimal model parameters for the generator.

Although training can be conducted using the target outputs  $Y$ , note that they are *not* available during test time. Therefore, the model uses its current prediction  $\hat{y}_t$  with hidden state  $\mathbf{h}_t$  to predict  $p(\hat{y}_{t+1} | \hat{y}_t, \mathbf{h}_t; \theta)$ . As already discussed, this can lead to an accumulation of errors. The current prediction  $\hat{y}_t$  can be obtained by either acting greedily and selecting the most probable word or chosen by sampling from the distribution  $p_\theta(y | \mathbf{h}_t; \theta)$ , calibrated using the `softmax` output.

### 3.3.1 Addressing Compounding Errors

Teacher forcing is when a sequence predictor is given full supervision during training, which is the most popular way to train NLMs. However, this does not reflect sequence generation at test time since targets are not provided and the model has to rely on its own past predictions to generate the next word. Scheduled Sampling (`SS`) addresses this by alternating between  $y_{t-1}$  and  $\hat{y}_{t-1}$  at training time to encourage the model to improve multi-step ahead prediction used at test time. The trade-off is controlled with probability  $\gamma_i$  ( $\in [0, 1]$ ) at the  $i$ -th epoch such that with probability  $(1 - \gamma_i)$  the model chooses  $\hat{y}_{t-1}$ . Herein, we denote the sampling rate as  $\epsilon$  for the schedule corresponding to `SS` and that for `NNRS` by  $\gamma$ . The  $\epsilon_i$  coefficients are incrementally decreased during training as a curriculum learning strategy where the model uses more true targets at the beginning of training and gradually changes to using its own predictions during learning.

There are three ways in which  $\epsilon_i$  is set, (1) a linear decay, (2) an exponential decay or (3) an inverse Sigmoid decay. However, as noted in previous work [177], `SS` has the limitation that it can lead to learning

an incorrect conditional distribution. Additionally, it is assumed that the teacher is optimal when using full supervision. Next, we describe our proposed method that addresses these criticisms while also being sufficiently modular to be used in conjunction with other sampling-based techniques for mitigating exposure bias.

### 3.3.1.1 Nearest-Neighbor Replacement Sampling

**DEFINING NEIGHBORS VIA EMBEDDING SIMILARITY:** Suppose we have  $d$ -dimensional pretrained embeddings for words  $w \in V$ , arranged as rows in an embedding matrix  $\mathbf{E} \in \mathbb{R}^{|V| \times d}$ . We denote the embedding of word  $w$  by  $\mathbf{w} \in \mathbb{R}^d$ . For each  $w \in V$ , we measure its similarity to all other words  $w' \in V - \{w\}$  using the cosine similarity,  $\cos(w, w')$ , given by (3.2).

$$\cos(w, w') = \frac{\mathbf{w}^\top \mathbf{w}'}{\|\mathbf{w}\| \|\mathbf{w}'\|} \quad (3.2)$$

We then define the neighborhood similarity matrix,  $\mathbf{N} \in \mathbb{R}^{|V| \times k}$ , containing similarities of the top- $k$  most similar words  $w'$  to  $w$  according to  $\cos(\mathbf{w}, \mathbf{w}')$  for all  $w \in V$ . We denote  $\mathbf{N}_w$  as the row of neighbor embedding similarities corresponding to word  $w$ . Once  $\mathbf{N}$  is computed, we convert each row to a probability distribution in order to sample the neighbors, which we denote as  $\tilde{\mathbf{N}}$  where for a given row corresponding to  $w$ ,  $\sum_{i=1}^k \tilde{\mathbf{N}}_{w,i} = 1$ . Hence, a neighbor  $w'$  in the  $i$ -th position of  $\tilde{\mathbf{N}}_{w,i}$  is associated with a probability  $p(w'; w, k)$  given by (3.3), where the sampling probabilities are defined using the `softmax` function to normalize the cosine similarities between pretrained word embeddings.

$$p(w'|w; k, \tau) = \frac{\exp(\cos(\mathbf{w}, \mathbf{w}')/\tau)}{\sum_{\mathbf{u} \in \mathbf{N}_w} \exp(\cos(\mathbf{u}, \mathbf{w})/\tau)} \quad (3.3)$$

Here the temperature  $\tau$  controls the “peakiness” of the distribution, lower  $\tau$  corresponding to much higher sampling probability for the closest neighbor and far smaller probabilities for the  $k$ -th furthest neighbor. Furthermore, a  $w$  that occurs at  $y_{t-1}$ , can instead be represented as a weighted average (i.e centroid) of its neighbor embeddings  $\tilde{\mathbf{w}}$  as shown in (3.4) and subsequently passed as input at  $x_t$ .

$$\tilde{\mathbf{w}} = \frac{1}{k} \sum_{i=1}^k p(w'_i|w, k) \mathbf{w}'_i \quad (3.4)$$

In our experiments,  $k \approx \log_2(|V|)$  which is sample efficient and speeds up sampling at run time when compared to using the entire vocabulary of size  $|V|$ . Note that, in the case where  $|V|$  is very large (e.g  $|V| > 10^4$ ), efficient k-NN search can be performed using KD-Trees, Metric Trees or Cover Trees algorithms [195], which has been applied in a similar

context [40]. In the best case, computation can be reduced from  $\mathcal{O}(n^2)$  to  $\mathcal{O}(n \log n)$  at the expense of some accuracy. However, this was not required as  $|V| \ll 10^4$  for the datasets we report results on.

During training, samples from top- $k$ -neighbors are drawn  $\tilde{w} \sim \tilde{\mathbf{N}}_w$ , while  $\tilde{\mathbf{N}}_w$  can change at the end of each epoch  $i \in \Gamma$  based on the validation perplexity  $\ell$  by updating  $\tau$ . This has the effect of shifting the probability mass for the  $k$ -neighbors sampling probability distribution, proportional to the increase or decrease in  $\ell$  from epoch  $i$  to  $i + 1$ .

At the start of training the model is conservative, only sampling the nearest neighbors. As the model begins to learn and minimize the loss on the validation set, it gradually explores more distant  $k$  neighbors. If replacing neighbors that have smaller cosine similarity has a similar effect on the validation loss to neighbors with higher cosine similarity, then  $\tau$  is increased and the model carries out more exploration. In the opposite case, where validation loss decreases,  $\tau$  becomes smaller and the model will only choose the closest of the  $k$ -neighbors for replacement. Concretely,  $\ell_i$  is the validation set loss at epoch  $i$ , and  $\ell_*$  is the best validation performance up until  $i$  number of epochs. When  $\ell_i < \ell_*$ ,  $\tau$  is increased using the update rule shown in (3.5) where  $\tau_i$  controls the temperature in (3.3) at epoch  $i$  and  $\varepsilon_{i-1} = (2^{\tau_{i-1}} - 1)$ .

$$\tau_i := \begin{cases} \tau_{i-1} + |\tau_{i-1} - \varepsilon_{i-1}|, & \ell_i - \ell_* \geq 0, \\ \tau_i - |\tau_i - \varepsilon_{i-1}|, & \ell_i - \ell_* < 0 \end{cases} \quad (3.5)$$

Because this procedure depends on  $\ell$ , the sampling process is directly controlled by the performance metric of interest.  $\tau \in [0.5, 10]$  is used to prevent (3.3) becoming too narrow or completely uniform.

This approach can be considered as an auxiliary task optimized for the expected reward. However, we require a curriculum for NNRS to control the amount of replacement sampling, as  $\tau$  would be increased very early on in training because large loss reductions are made where the model is initialized at random. Thus, changes in  $\tau$  have a larger effect on the overall sampling process as the chosen schedule monotonically increases the replacement sampling rate over training epochs.

(3.6) shows the procedure for choosing whether to use past target  $y_{t-1}$ , past prediction  $\hat{y}_{t-1}$  or a sampled neighbor of  $y_{t-1}$  and assign it to the next input  $x_t$ .

$$x_t = \begin{cases} \tilde{y}_{t-1} \sim \mathbb{B}(\tilde{\mathbf{N}}_y, \hat{y}), & (\epsilon > \xi_{ss}) \ \& \ (\gamma > \xi_{nnrs}) \\ \hat{y}_{t-1}, & (\epsilon > \xi_{ss}) \ \& \ (\gamma < \xi_{nnrs}) \\ y_{t-1}, & (\epsilon < \xi_{ss}) \ \& \ (\gamma > \xi_{nnrs}) \end{cases} \quad (3.6)$$

We start by sampling uniformly at random  $\xi_{ss} \sim \text{Uniform}(0, 1)$  and  $\xi_{nnrs} \sim \text{Uniform}(0, 1)$  for each sample  $Y$  within a training mini-batch  $B^i$

in  $\mathbf{B}_{tr} = \{B^1, B^2, B^i, \dots, B^K\}$ . Here,  $\xi_{ss}$  is used as a threshold to decide which  $y \in Y$  are used for **SS** and  $\xi_{nnrs}$  sets a threshold for **NNRS** when both are used in conjunction. When both  $\epsilon$  and  $\gamma$  choose the same tokens for sampling, we choose one at random with equal probability. This point becomes more relevant towards the end of training when the schedule outputs a high sampling rate. For faster inference, we fix the sampled indices corresponding to  $y \in Y$  for all samples in  $Y$ . This means **NNRS** can be performed in parallel on all mini-batch samples in  $\mathbf{B}_{tr}$ . Below, we have described the use of **NNRS** in conjunction with another popular sample-based method to demonstrate its modularity. However, **NNRS** can also be used standalone as we see later in our experiments. In fact, **NNRS** can be expanded upon to learn which neighbors to sample as we describe next.

**GUMBEL SOFTMAX NEIGHBOR SAMPLING:** An alternative to updating the temperature via the incremental update rule in (3.6) is to use a straight-through estimator that allows us to differentiate through drawn samples  $p(w'|w; k, \tau)$ . This can be achieved using the Gumbel-Softmax [GS; 183] trick. For **NNRS**, we refer to this as Gumbel-Softmax Neighbor Sampling (GSNS). The GS allows us to sample and backpropagate through  $\tilde{\mathbf{N}}$  by reparameterizing the sampling process as  $\mathbf{d}\tilde{\mathbf{N}}_w / \mathbf{d}\alpha$  where  $\alpha$  is a multinomial distribution. Equation 3.7 shows the GS where each componentwise Gumbel noise  $\kappa \in [1.., k]$  added to the original distribution  $p(\tilde{\mathbf{N}}_w)$  for  $w$ , we find  $\kappa$  that maximizes  $G_w^\kappa := \log \alpha_w^\kappa - \log(-\log U_w^\kappa)$  and then set  $D_w^\kappa = 1$  and the remaining elements  $D_w^{-\kappa} = 0$  (i.e a one-hot vector). Here,  $U_w^\kappa \sim \text{Uniform}(0, 1)$  and  $\alpha_w^\kappa$  is drawn from the discrete distribution  $D \sim \text{Discrete}_w(\alpha)$ . We then sample  $p(w'|w; k, \tau)$  as in Equation 3.7 after updating  $\nabla_{p(\mathbf{N}_w)} \ell_i$ .

$$p(w'|w; k, \tau) = \frac{\exp((\log \alpha_w^\kappa + G_w^\kappa) / \tau)}{\sum_{i=1}^k \exp((\log \alpha_w^i + G_w^i) / \tau)} \quad (3.7)$$

This allows us to draw samples from a Gumbel distribution while performing gradient updates on  $p(\mathbf{N}_w)$ . In contrast to **NNRS**, which performs updates according to the validation perplexity at the end of the of each training epoch, we instead update the neighbor distribution throughout training. However, similar to before, the curriculum still controls the amount of GSNS updates that are performed. Therefore the gradient updates are only for the tokens, which are chosen for GSNS according to the curriculum schedule. We test  $\tau = 0.5$  as a constant during training, We set an upper bound of  $\tau = 10$ , which corresponds to a heavy dampening of (3.7). Small  $\tau$  early avoids high variance in gradient updates, even-though we would expect the curriculum to use **NNRS** less early on.

We expect the output to be less sensitive to perturbations in the input because the input is locally bounded by the space occupied by the  $k$  nearest neighbors of the target  $\tilde{y}_t$ . Likewise,  $\tilde{y}_t$  can be considered

as emulating the problem of compounding errors, since the conditional probability  $p(y_{t+1}|x_{1:t}, \tilde{y}_t; \theta)$  is conditioned on the sampled neighbor  $\tilde{y}_t$  instead of the true target  $y_t$ . In cases where the model finds it difficult to transition from using  $y_{t-1} \rightarrow \hat{y}_{t-1}$ , interpolating with the neighborhood samples  $\tilde{y}$  can provide a smoother policy ( $y \rightarrow \tilde{y} \rightarrow \hat{y}$ ). This smoothing assigns some mass to unseen transitions (similar to Laplacian smoothing), bounded by  $k$  neighbors, which is directly proportional to transition probabilities. We now consider curriculum schedules to monotonically increase both  $\gamma$  and  $\gamma_{\text{nnrs}}$ , corresponding to the sampling rates for **SS** and **NNRS** respectively and aim to identify schedules that help mitigate compounding errors by controlling the amount of exploration of neighbors throughout training.

**Objective Definition** We note that, in **SS**, the objective is not a strictly proper scoring rule (i.e the maximum of the function is unique) since the objective is dependent on the model's own distribution. However, in **NNRS**, we do not use the model's distribution but instead neighbors of past targets which are not subject to change throughout training. In other words, the sampling procedure is chosen by a predefined curriculum and therefore is independent of the model, thus the maximum of this scoring function remains unique.

In the case of a static sampling rate for  $\epsilon$  and  $\gamma$ , we can define the log-likelihood loss with **SS** and **NNRS** in terms of KL-divergences, shown in (3.8).  $P_{x_t}$  and  $Q_{x_t}$  are the marginal distributions for input token  $x_t$ , while  $P_{x_t|x_{t-1}=h}$  and  $Q_{x_t|x_{t-1}=h}$  are the conditional distributions. SGD with cosine annealed learning rate [354] is also considered for optimizing the model.

$$\begin{aligned}
D_{\text{SS-NNRS}}[P||Q] &= \text{KL}[P_{x_{t-1}}||Q_{x_{t-1}}] + \\
&\underbrace{(1 - \epsilon)\mathbb{E}_{h \sim Q_{x_{t-1}}} \text{KL}[P_{x_t|x_{t-1}=h}||Q_{x_t|x_{t-1}=h}]}_{\text{SS}} + \\
&\underbrace{\epsilon \mathbb{E}_{h \sim P_{x_{t-1}}} \text{KL}[P_{x_t|x_{t-1}}||Q_{x_t|x_{t-1}}]}_{\text{SS}} + \\
&\underbrace{(1 - \gamma)\mathbb{E}_{h \sim Q_{x_{t-1}}} \text{KL}[P_{x_{t+1}}||Q_{x_t|x_{t-1}=h}]}_{\text{k-NN replacement sampling}} + \\
&\underbrace{\gamma \mathbb{E}_{h \sim P_{x_{t-1}}} \text{KL}[P_{x_t|x_{t-1}}||Q_{x_t|x_{t-1}}]}_{\text{k-NN replacement sampling}}
\end{aligned} \tag{3.8}$$

In practice, we only need to use the CE loss when using **NNRS**. The aforementioned KL analysis suggests that when using the CE loss, we the scoring function is proper, unlike the **SS** objective. At this point, we have defined the whole process to carry out **NNRS** and summarize the training with **NNRS** in Algorithm 1.

**Algorithm 1** NNRS-based Training.

---

```

1: input: Training epochs  $\Gamma$ . Sentence length  $T$  for training mini-batch
   training data  $\mathbf{B}_{\text{tr}} = \{B_{\text{tr}}^1, B_{\text{tr}}^2, \dots, B_{\text{tr}}^K\}$  and validation data  $\mathbf{B}_{\text{val}} =$ 
    $\{B_{\text{val}}^1, B_{\text{val}}^2, \dots, B_{\text{val}}^R\}$ . An RNN  $f_{\theta}(\cdot, \cdot)$  and cur. schedules  $g(\cdot)_{\text{ss}}, g(\cdot)_{\text{nnrs}}$ .
2: Initialize the RNN parameters  $\theta$  at random Define k-NN embedding
   similarity for vocabulary  $V$  using (3.2) and normalize to a probability
   distribution  $p(\tilde{\mathbf{N}})$  using (3.3)
3: Set  $\tau = 0.1, i = 0, \xi = 0, \ell_* = |V|$ 
4: for  $B^k \in \mathbf{B}_{\text{tr}}$  do
5:   for step  $t = 1 \rightarrow T_k$  do
6:     Set  $\epsilon = g_{\text{ss}}(i), \gamma_{\text{nnrs}} = g_{\text{nnrs}}(i), \ell_i = 0$ 
7:     # Bernoulli sample for NNRS and SS rates
8:     Sample  $\xi_{\text{ss}} \sim \mathbf{U}(0, 1)$  and  $\xi_{\text{nnrs}} \sim \mathbf{U}(0, 1)$ 
9:     # Contrastive mixup representations
10:    Sample  $\mathbf{x}_t$  using (3.6) and input  $\hat{\mathbf{y}}_t, h_t = f_{\theta}(\mathbf{x}_t, \mathbf{h}_{t-1})$ 
11:  end for
12:  # Compute cross-entropy loss
13:   $\ell_{\text{tr}} = \mathcal{L}_{\text{ce}}(\hat{\mathbf{Y}}, \mathbf{Y})$ 
14:  # Update parameters
15:   $\theta := \eta\theta + (1 - \eta)\nabla_{\theta} \log \ell_{\text{tr}}$ 
16:  Compute  $\ell_i$  by repeating above on  $\mathbf{B}_{\text{val}}$  without updating  $\theta$ 
17:  Update  $\ell_*$  if  $\ell_i < \ell_*$ 
18:  if using the Gumbel-Softmax
19:    Update  $\mathbf{N} := \beta\mathbf{N} - (1 - \beta)\nabla_{\tilde{\mathbf{N}}} \ell_*$ 
20:  else
21:    Update  $\mathbf{N}$  by changing  $\tau$  based on the difference between  $\ell_*$  and
     $\ell_i$  using (3.5)
22:  Renormalize  $p(\mathbf{N})$  using the softmax
23: end for

```

---

3.3.1.2 *Experiments*

We use the two well-established language modeling datasets for our experiments, Penn-Treebank [(PTB) 272] and WikiText-2 [43].

3.3.2 *Experimental Results*

Figure 3.1 shows the perplexity scores on WikiText-2 for the different parameter settings for  $\epsilon$  and  $\gamma$  over 40 training epochs. Solid lines indicate validation perplexity scores throughout training and dashed horizontal lines indicate corresponding test perplexity score. We find that most learning is carried out after 20 epochs given that the learning rate is initially high ( $\alpha = 20$ ) and annealed using cosine annealing. Based on the validation set performance, we found the optimal settings to be  $\epsilon = 0.5$  and  $\gamma = 0.2$  with a static probability sampling rate. Additionally, using an exponential sampling rate outperforms the linear and sigmoid functions. It appears that this is because the exponential function carries out the majority of sampling late in the training when the model has reached learning capacity from the teacher policy.

Configuration	Parameter Setting				Penn-Treebank							
	$\epsilon_s$	$\epsilon_e$	$\gamma_s$	$\gamma_e$	Linear		S-Shaped Curve		Exponential Increase		Static	
					Valid	Test	Valid	Test	Valid	Test	Valid	Test
No Sampling	-	-	-	-	76.25	71.81	76.25	71.81	76.25	71.81	76.25	71.81
TPRS-1	0	0	0	0.2	<b>81.42</b>	<b>76.58</b>	<b>83.40</b>	<b>81.22</b>	77.56	73.02	<b>81.45</b>	<b>80.36</b>
TPRS-2	0	0	0	0.3	96.91	94.30	88.79	86.17	<b>77.63</b>	<b>72.80</b>	91.73	84.48
TPRS-3	0	0	0	0.5	97.62	94.78	88.46	86.05	<b>76.60</b>	72.77	98.31	95.23
NNRS-1	0	0	0	0.2	<b>80.91</b>	<b>76.70</b>	<b>83.17</b>	<b>79.62</b>	<b>76.00</b>	<b>72.19</b>	<b>82.23</b>	<b>80.03</b>
NNRS-2	0	0	0	0.3	96.66	93.18	88.68	85.29	77.12	73.05	92.12	84.29
NNRS-3	0	0	0	0.5	96.95	93.15	88.21	84.47	75.91	72.46	97.55	94.79
SS-1	0	0.2	0	0	83.63	80.03	<b>82.33</b>	<b>79.02</b>	76.71	73.32	74.50	70.20
SS-2	0	0.3	0	<b>0.4</b>	<b>79.82</b>	84.42	80.03	76.43	73.46	<b>74.56</b>	<b>70.25</b>	
SS-3	0	0.5	0	0	92.48	88.92	85.87	82.37	76.41	73.15	74.11	69.48
SS-4	0	0.8	0	0	92.94	88.85	87.29	84.27	<b>76.22</b>	<b>72.98</b>	74.02	70.23
SS-NNRS-1	0	0.2	0	0.2	<b>83.62</b>	<b>80.03</b>	<b>81.99</b>	<b>77.86</b>	76.71	73.31	74.50	70.20
SS-NNRS-2	0	0.3	0	0.3	95.39	92.18	87.97	84.64	75.43	72.46	74.64	70.43
SS-NNRS-3	0	0.5	0	0.2	95.90	92.96	88.56	84.96	<b>74.83</b>	<b>70.95</b>	<b>72.89</b>	<b>69.06</b>
SS-NNRS-4	0.2	0.5	0.2	0.5	96.78	93.20	88.56	87.06	76.41	73.15	74.42	69.88
SS-NNRS-5	0	0.5	0	0.5	97.12	93.90	90.37	86.25	76.02	72.31	74.08	70.79
SS-NNRS-6	0	0.8	0	0.2	95.91	92.87	81.89	78.54	74.74	70.64	74.02	70.23
SS-NNRS-7	0.2	0.8	0.2	0.5	96.77	93.64	88.27	85.04	<b>76.35</b>	<b>72.74</b>	74.43	70.12

Table 3.1: Test Perplexity for a 2-hidden layer LSTM [165] using Transition Probability Replacement Sampling (TPRS), SS and Nearest-Neighbor Replacement Sampling (NNRS) with linear, s-shaped curve and exponential sampling functions.

For PTB, static and exponential settings for both SS and NNRS show best performance and converge quicker. In all cases, convergence for all functions behaves as expected for  $\{\sigma, \gamma\} < 0.3$ . We also see the same trend as with WikiText-2, where the exponential function allows for higher sampling rates when compared to linear and sigmoid functions. Again, this suggests that NNRS is most effective near convergence, as the sampling probability exponentially increases while the validation perplexity begin to plateau over epochs.

SCHEDULE PARAMETER GRID SEARCH RESULTS Table 3.1 and Table 3.2 shows the results of the model with varying  $\epsilon$  and  $\gamma$  upper and lower thresholds using a linear, s-curve, exponential and static sampling strategy for all tested datasets. Here, subscript  $s$  for  $\epsilon_s, \gamma_s$  are the initial probabilities at  $i = 0$  and subscript  $e$  for  $\epsilon_e, \gamma_e$  denote end probabilities  $i = \Gamma$ . Bolded results are those which perform within the cell that are contained (aggregated by configuration and type of schedule e.g SS-1 - SS4 for linear) and shaded cells corresponds to the best performing configuration and schedule for either Wikitext-2 or PTB. These experiments describe results using the simple update rule described in Equation (3.5).

NNRS and SS used in conjunction yield the best performance in most cases for both datasets. Best performance for both Wikitext-2 and PTB is found with  $\epsilon \in [0, 0.5]$  and  $\gamma \in [0, 0.2]$ , and slightly improves over only using SS. For PTB,  $\gamma_e = 0.2$  performs the best for linear and sigmoid functions,  $\gamma = 0.5$  for exponential and static sampling rates and overall

Configuration	Parameter Setting				Wiki-102							
	$\epsilon_s$	$\epsilon_e$	$\gamma_s$	$\gamma_e$	Linear		S-Shaped Curve		Exponential Increase		Static	
					Valid	Test	Valid	Test	Valid	Test	Valid	Test
No Sampling	-	-	-	-	140.28	128.78	140.28	128.78	140.28	128.78	140.28	128.78
TPRS-1	0	0	0	0.2	<b>143.78</b>	<b>131.18</b>	<b>137.69</b>	<b>127.05</b>	137.63	136.88	136.31	126.49
TPRS-2	0	0	0	0.3	154.93	144.02	146.92	137.07	<b>137.11</b>	<b>125.41</b>	<b>137.10</b>	<b>125.95</b>
TPRS-3	0	0	0	0.5	159.11	148.41	148.10	139.58	138.46	127.97	138.53	129.87
NNRS-1	0	0	0	0.2	<b>142.53</b>	<b>130.45</b>	<b>137.08</b>	<b>126.82</b>	136.81	136.11	135.06	136.02
NNRS-2	0	0	0	0.3	154.50	143.13	149.69	136.98	<b>136.83</b>	<b>125.53</b>	136.34	125.84
NNRS-3	0	0	0	0.5	158.30	147.13	148.02	137.34	137.56	127.61	<b>132.62</b>	<b>121.50</b>
SS-1	0	0.2	0	0	139.31	128.50	143.82	131.08	137.72	126.46	135.55	122.61
SS-2	0	0.3	0	0	137.78	126.00	138.39	126.80	<b>135.89</b>	<b>125.03</b>	131.29	121.52
SS-3	0	0.5	0	0	<b>135.14</b>	<b>124.29</b>	<b>136.88</b>	<b>125.41</b>	136.98	125.96	<b>131.28</b>	<b>121.51</b>
SS-4	0	0.8	0	0	140.97	130.00	141.13	129.99	138.09	127.23	134.32	122.86
SS-NNRS-1	0	0.2	0	0.2	139.32	128.50	<b>141.57</b>	<b>129.67</b>	137.73	126.47	135.55	122.60
SS-NNRS-2	0	0.3	0	0.3	137.78	126.01	147.34	135.61	136.02	125.73	137.73	126.47
SS-NNRS-3	0	0.5	0	0.2	<b>134.79</b>	<b>123.90</b>	149.00	137.97	<b>135.82</b>	<b>124.72</b>	<b>130.95</b>	<b>120.76</b>
SS-NNRS-4	0.2	0.5	0.2	0.5	150.97	138.59	146.01	134.67	135.88	126.02	123.84	121.98
SS-NNRS-5	0	0.5	0	0.5	136.22	125.70	151.39	138.44	137.02	125.84	132.17	121.86
SS-NNRS-6	0	0.8	0	0.2	148.96	130.01	141.14	129.99	138.09	127.24	134.32	122.89
SS-NNRS-7	0.2	0.8	0.2	0.5	155.17	148.21	149.58	137.83	135.45	124.82	131.09	121.43

Table 3.2: Test Perplexity for a 2-hidden layer LSTM [165] using Transition Probability Sampling (TPRS), SS and Nearest-Neighbor Replacement Sampling (NNRS) with linear, s-shaped curve and exponential sampling functions.

Eval.	Model	MLE		TPRS		NNRS		SS		SS-NNRS	
BLEU-4	LSTM	6.11	6.62	5.74	5.31	<b>4.49</b>	<b>4.31</b>	6.02	5.66	5.03	4.79
	GRU	6.03	6.31	5.45	5.03	4.79	4.67	5.73	5.50	5.26	4.61
	Highway	6.84	7.04	6.34	6.13	5.77	5.59	5.63	5.10	6.08	5.28
WMD	LSTM	0.51	0.46	0.44	0.36	0.41	0.36	0.43	0.42	<b>0.36</b>	<b>0.34</b>
	GRU	0.53	0.49	0.43	0.34	0.42	0.38	0.40	0.39	<i>0.38</i>	<i>0.35</i>
	Highway	0.36	0.53	0.51	0.48	0.40	0.39	0.39	0.42	0.40	<i>0.40</i>

Table 3.3: PTB Self-BLEU4 and Self-WMD Validation/Test scores

a constant sampling rate. Although, the exponential schedule is better than the sigmoid and linear scheduled for both  $\epsilon$  and  $\gamma$  on both datasets. It seems that this is because the majority of neighbor replacements are towards the end of the training when the learner has already reached the full capacity in what can be learned from the teacher policy. This coincides with the theoretical guarantees of using an exponential decay schedule provided in DAgger [362].

Overall, interpolating between NNRS and SS produces the best performance on average for both datasets. A low constant sampling rate yields best performance for both datasets. Controlling the temperature  $\tau$  based on the validation perplexity allows all  $k$ -neighbors to be explored. When compared to using no sampling, there is an 8 point perplexity decrease using our approach on WikiText-2 and a 2.75 test perplexity

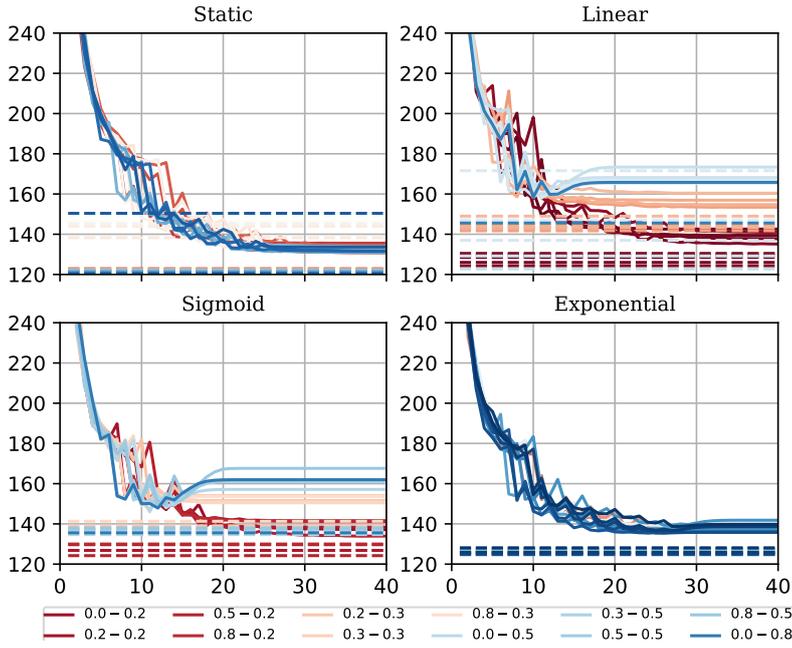


Figure 3.1: Perplexity on WikiText2: SS and NNRS

Eval.	Model	MLE		TPRS		NNRS		SS		SS-NNRS	
BLEU-4	LSTM	7.84	8.75	7.14	6.88	7.38	7.83	10.06	9.61	7.72	7.88
	GRU	8.22	9.53	7.33	7.41	<b>7.88</b>	<b>7.23</b>	9.78	9.90	7.27	7.50
	Highway	9.13	9.26	8.04	7.93	8.29	9.14	11.11	10.51	8.73	8.18
WMD	LSTM	0.41	0.40	0.48	0.41	0.35	0.30	0.34	0.32	<b>0.29</b>	<b>0.28</b>
	GRU	0.43	0.42	0.47	0.36	0.38	0.34	0.39	0.33	0.36	0.31
	Highway	0.48	0.45	0.51	0.53	0.52	0.54	0.59	0.31	0.36	0.36

Table 3.4: WikiText-2 Self-BLEU4 and Self-WMD Validation/Test scores

decrease on PTB. In comparison to Transition Probability Replacement Sampling (TPRS), **NNRS** slightly improves while requiring less memory to store embeddings.

### 3.3.3 Evaluation

Apart from improving over baselines in terms of perplexity, we identify what tradeoff is incurred between text generation quality and diversity for our proposed method. In these experiments we use the best performing  $\epsilon$  and  $\gamma$  settings found post-analysis for MLE ( $\gamma_s = \gamma_e = 0$  and  $\epsilon_s = \epsilon_e = 0$ ), **SS** ( $\epsilon_s = 0, \epsilon_e = 0.5$ ), TPRS, **NNRS** ( $\gamma_s = 0, \gamma_e = 0.2$  with a static sampling rate, for **SS**  $\epsilon_s = 0, \epsilon_e = 0.5$ ) and **SS-NNRS** (combine previous  $\gamma$  and  $\epsilon$  settings) trained models.

	Model	MLE		TPRS		NNRS		SS		SS-NNRS	
BLEU4	LSTM	7.87	8.28	9.24	8.16	11.81	11.26	10.93	10.53	<b>11.62</b>	<b>11.20</b>
	GRU	9.39	8.58	9.49	10.67	11.35	11.04	10.98	11.60	12.14	11.79
	Highway	9.03	8.56	9.72	9.40	10.81	10.37	11.24	11.96	<b>13.75</b>	<b>14.03</b>
WMD	LSTM	0.72	0.84	0.85	0.93	0.91	0.88	0.89	0.91	<b>0.95</b>	<b>0.93</b>
	GRU	0.72	0.72	0.67	0.63	0.70	0.69	0.70	0.69	<i>0.82</i>	<i>0.84</i>
	Highway	0.70	0.69	0.74	0.72	0.78	0.76	0.72	0.75	<i>0.79</i>	<i>0.80</i>

Table 3.5: WikiText-2 BLEU-4 &amp; WMD Quality Scores

Table 3.3 and Table 3.4 shows the self-BLEU [496] scores for PTB and WikiText-2 respectively, where higher scores correspond to less diversity in the predictions. Similarly, we also define a self-WMD to measure the semantic diversity which is computed by averaging the average Word Movers Distance [WMD; 212] between  $\ell_2$ -normalized embeddings (the same GoogleNews pretrained skipgram embeddings [283] that we used to define neighbors) of predicted words and target words in the test set. The WMD scores are in  $[-1, 1]$  as the cosine similarity is used as the WMD measure. After computing WMD, the scores are normalized to be in the range of  $[0, 1]$ . Hence, low self-WMD similarity scores closer to 0 correspond to high diversity. In contrast, when both self-BLEU and self-WMD are high in Table 3.5, this signifies high quality. We note that we only consider terms in the validation and test sets that also occur in the training set as including them in the evaluation deflates quality scores.

TEXT GENERATION DIVERSITY Zhu et al. [496] measured the diversity of text generation by computing BLEU scores between a collection of generated samples (referred to as self-BLEU). Self-BLEU can help identify mode collapse in generative models (e.g generative adversarial networks [128]). In the context of unconditional and conditional NLM, this can correspond to only predicting a subset of the vocabulary whereby the model fails to predict other *modes* (i.e words) which may occur due to the infrequent use of some terms in the long tail of the frequency distribution. In Table 3.3 and Table 3.4, we calculate Self-BLEU4 on both the validation and test sets for the best performing models on PTB and WikiText-2 respectively. To speed up computation, we avoid computing the self-BLEU4 between all generated samples but instead within each mini-batch and then average over all mini-batches. We find that self-WMD in particular shows an increase in diversity for NNRS when compared to standard ML training. Although coherently evaluating diversity is difficult, it at least suggests that NNRS is generating sentences that are semantically similar when using the aforementioned GoogleNews pretrained skipgram embeddings.

**TEXT GENERATION QUALITY** Following prior work by Yu et al. [473], we compute BLEU between the predictions and the corresponding targets as shown in Table 3.5. Similar to the diversity evaluation, WMD is computed between the pretrained embeddings of the predicted text and target text. We find that SS-NNRS leads to slightly more diverse sentences given BLEU-4, but when taking sentence-level semantic similarity using WMD, we find that the improvement is more evident as WMD takes into account the similarity between  $k$  local neighbors and targets in the NNRS method, unlike BLEU-4 which treats all incorrect predictions as equally bad. For larger  $k$ , diversity is increased in expense for generation quality, while  $k=0$  corresponds to ML training.

### 3.3.4 Summary

To summarize, our findings based on the above LM experiments on PTB and WikiText-2 can be described as follows.

Firstly, SS-NNRS with  $\epsilon = 0 \rightarrow 0.5$  and  $\gamma = 0 \rightarrow 0.2$  has shown the best performance on both PTB and WikiText-2. Hence, standard ML training can be used early on in training to allow the model to be robustness enough before learning from synonymous neighboring tokens. Secondly, NNRS improves over SS when used in isolation, most notably using a linear schedule. One posits that using past predictions in the intermediate stages of training can be too difficult for the model to adapt to since the model predictions early on are poor and hence recorrecting EB is more difficult. Learning from the targets neighbors is an easier alternative before beginning to increase the sampling rate for choosing past predictions.

Thirdly, sampling neighbors using the whole transition probability matrix leads to poor results even with relatively low sampling rates. Therefore, not only are lower-dimensional embeddings more efficient in sampling during training, but also leads to better performance in the practical setting with a set number of epochs. This related to the tradeoff between exploration and exploitation, where Transition Probability Replacement Sampling (TPRS) exploration is too much and therefore leads to a decrease in performance, as reflected in Table 3.2.

Lastly, larger test perplexity reductions are found for WikiText-2 when using NNRS. Unlike PTB which has truncated vocabulary size of 10,000, WikiText-2 does not have a reduced vocabulary set and therefore rare terms are kept in the vocabulary, which contains 50,000 words. Hence, NNRS is particularly useful for improving the predictions of less frequent words, particularly when the dataset is relatively small which leads to poor approximations of the transition probability matrix that is necessary when using TPRS.

In the next chapter, we describe a method that approximates the decoder and softmax normalization of NLMs, which can be used in

conjunction with methods that mitigate exposure bias, such as NNRS or SS that has been discussed in this chapter.

# 4

---

## MODEL COMPRESSION VIA ERROR-CORRECTING CODES

---

In the previous chapter we discussed a method that mitigates exposure bias and therefore improves training efficiency in Neural Sequence Predictors (NSPs). In this chapter, we discuss an approach to compactly represent outputs to overcome the computational bottlenecks associated with very high dimensional output spaces, specifically in neural sequence prediction. This too is a source of slower training times.

Namely, *error-correcting output codes* are proposed to avoid exact softmax normalization to allow for a tradeoff between speed and performance. Instead of minimizing measures between the predicted probability distribution and the true distribution, error-correcting codes are used to represent both predictions and outputs and perform maximum likelihood over a factorial distribution.

Secondly, multiple ways to improve accuracy and convergence rates are proposed to maximize the separability between codes that correspond to classes proportional to word embedding similarities. This semantically ordered codebook is used in the main contribution, *Latent Variable Mixture Sampling* - a technique that is used to mitigate exposure bias, which can be integrated into training latent variable-based NSPs such as ECOC. This involves mixing the latent codes of past predictions and past targets in one of two ways: (1) according to a predefined sampling schedule or (2) a differentiable sampling procedure whereby the mixing probability is learned throughout training by replacing the greedy argmax operation with a smooth approximation. ECOC-NSP leads to consistent improvements on LM datasets and the proposed Latent Variable mixture sampling methods are found to perform well for text generation tasks such as image captioning.

### 4.1 INTRODUCTION

Sequence Modeling (SM) is a fundamental task in natural language which requires a parametric model to generate tokens given past tokens. SM underlies many types of structured prediction tasks in natural language, such as LM, Sequence Tagging (e.g NER, Constituency/Depen-

dency Parsing) and Text Generation (e.g Image Captioning and Machine Answering [404]). The goal is to learn a joint probability distribution for a sequence of length  $T$  containing words from a vocabulary  $\mathcal{V}$ . This distribution can be decomposed into the conditional distributions of current tokens given past tokens using the chain rule shown in Equation 4.1. In Neural Sequence Prediction (NSP), an RNN  $f_\theta(\cdot)$  parameterized by  $\theta$  is used to encode the information at each timestep  $t$  into a hidden state vector  $h_t^l$  which is followed by a decoder  $z_t^l = h_t^l W^l + b^l$  and a normalization function  $\phi(z_t^l)$  which forms a probability distribution  $\hat{p}_\theta(y_t|x_t, h_{t-1})$ ,  $\forall t \in [0, 1, ..T]$ .

$$P(w_1, \dots, w_T) = \prod_{t=1}^T P(w_t|w_{t-1}, \dots, w_1) \quad (4.1)$$

However, training can be slow when  $|\mathcal{V}|$  is large while also leaving a large memory footprint for the respective input embedding matrices. Conversely, in cases where the decoder is limited by an information bottleneck, the opposite is required where more degrees of freedom are necessary to alleviate information loss in the decoder bottleneck. Both scenarios correspond to a trade-off between computation complexity and out-of-sample performance. Hence, it is desired that a newly proposed model has the property that the decoder can be easily configured to deal with this trade-off in a principled way.

ECOC-NSP is also applied in the context of exposure bias, as discussed in Chapter 3, whereby either predicted error codes or ground truth error-codes are Bernoulli sampled and passed as input at the next time-step.

Hence, error-correcting output code (ECOC) based NSP (ECOC-NSP) is proposed and addresses this desideratum. When given sufficient error codes ( $|\mathcal{V}| \gg |c| \gg \log_2(|\mathcal{V}|)$ ) while the codeword dimensionality  $|c| < |\mathcal{V}|$ , accuracy can be maintained compared to using the full softmax. Lastly, we show that this latent variable-based NSP approach can be extended to mitigate the aforementioned problem of compounding errors by using *Latent Variable Mixture Sampling* (LVMS). LVMS in an ECOC-NSP model also outperforms an equivalent Hierarchical Softmax-based NSP that uses Scheduled Sampling [31] and other closely related baselines. This is the first report of mitigating compounding errors when approximating the posterior in recurrent neural networks (RNNs).

**CONTRIBUTIONS** Our main contribution is an ECOC neural language model that requires less parameters than its softmax-based SM counterpart given sufficient separability between classes via error-checks. To achieve well-separated codes we also propose to rank order the codebook using pretrained embedding similarity where the number of error-correcting codes assigned to a token in the codebook is proportional to the cosine similarity between the tokens corresponding

pretrained word embedding and the most frequent tokens word embedding. Thirdly, to mitigate EB for latent variable approaches (e.g. ECOC approximations), we propose *Latent-Mixture Sampling*. This is then extended to *Differentiable Latent Variable Mixture Sampling* that uses the Gumbel-Softmax so that discrete categorical variables can be backpropagated through. We show that this performs comparably to other sampling-based approaches. Lastly, novel baselines such as Scheduled Sampling Hierarchical Softmax (SS-HS) and Scheduled Sampling Adaptive Softmax (SS-AS) are introduced in the evaluation of ECOC.

## 4.2 BACKGROUND

**ERROR-CORRECTING CODES** Error-Correcting Codes [140] were first applied in the context of solid-state electronics, soon followed by binary codes that were used in the NETtalk system [380]. Here, each class index was represented by a binary code  $C$  and a predicted code as  $\hat{C}$  from some parametric model  $f_\theta(\cdot)$ . Both error-correction bits and class bits make up a *codeword*  $C \in \mathcal{C}$ . When  $|\mathcal{V}| \neq 2^n$ , the remaining codes are used as error-correction bits  $k = |\mathcal{V}| - 2^n$ . This tolerance can be used to account for the information loss due to the sample size by measuring the distance (e.g. Hamming) between the predicted codeword and the true codeword with  $d$  error-correction bits. If the minimum distance between codewords is  $d$  then at least  $(d-1)/2$  bits can be corrected for and hence, if the Hamming distance  $d \leq (d-1)/2$  we will still retrieve the correct codeword. In contrast to using one bit per  $k$  classes in standard multi-class classification, error-correction cannot be achieved.

**WHY LATENT CODES FOR NEURAL SEQUENCE PREDICTION?** Targets are represented as 1-hot vectors (i.e. Kronecker delta) in standard training of NSPs, treated as a 1-vs-rest multi-class classification. This approach can be considered a special case of ECOC classification where the codebook  $\mathcal{C}$  with  $n$  classes is represented by an identity  $\mathbb{I}^{n \times n}$ . ECOC classification is well suited over explicitly using observed variables when the output space is structured. Flat-classification (1-vs-rest) ignores the dependencies in the outputs, in comparison to using latent codes that share some common latent variables between associative words. For example, in the former case, if we train a model that only observes the word “silver” in a sequence “...silver car..” and then at test-time observes “silver-back”, because there is high association between “silver” and “car”, the model is more likely to predict “car” instead of “gorilla”. ECOC is less prone to such mistakes because although a/some bit/s may be different between the latent codes for “car” and “gorilla”, the potential misclassifications can be re-corrected with the error-correcting bits. In other words, latent coding can reduce the variance of each

individual classifier and has some tolerance to mistakes induced by sparse transitions, proportional to the number of error-checks used.

**METHODS FOR SOFTMAX APPROXIMATION** Morin and Bengio proposed a **Hierarchical Softmax** (HS) that outputs short codes representing marginals of the conditional distribution, where the product of these marginals along a path in the tree approximate the conditional distribution. This speeds up training proportional to the traversed binary tree path lengths, where intermediate nodes assign relative probabilities of child nodes. Defining a good tree structure improves performance since semantically similar words have a shorter path and therefore similar representations are learned for similar words. HS and ECOC are similar insofar as they both can be interpreted as approximating the posterior as a product of marginal probabilities using short codes. However, ECOC tolerates errors in code prediction. This becomes more important as the code length (i.e the tree depth in HS) grows since the likelihood of mistakes becomes higher. Hence, ECOC can be considered a flexible tradeoff between the strictness of HS and the full softmax. The **Differentiated Softmax** (DS) uses a sparse linear block of weights for the decoder where a set of partitions are made according to the unigram distribution, where the number of weights are assigned proportional to the term frequency. This is intuitive since rare words require less degrees of freedom to account for the little amount of contexts in which they appear, compared to common words. The optimal spacing of error-checking bits between codewords corresponding to tokens is similar to varying branching sizes for different areas of the tree depending on the unigram frequency or allocating weights proportional to the frequency akin to Differentiable Softmax [59]. We also consider achieving the spacing via term frequency, but also use our proposed rank ordered word embedding cosine similarities as mentioned in Equation 4.3.1.

The **Adaptive Softmax** (AS) [132] provide an approximate hierarchical model that directly accounts for the computation time of matrix multiplications. AS results in 2x-10x speedups when compared to the standard softmax, dependent on the size of the corpus and vocabulary. They find on sufficiently large corpora (e.g Text8), accuracy is maintained while reducing the computation time.

**RECENT APPLICATIONS OF LATENT CODES** Shu and Nakayama recently used compositional codes for word embeddings to cut down on memory requirements in mobile devices. Instead of using binary coding, they achieve word embedding compression using multi-codebook quantization. Each bit  $c \in C$  comprises of a discrete code (0-9) and therefore at minimum  $\log_{10}(k)$  bits are required. They also propose to use the Gumbel-Softmax trick but for the purposes of learning the discrete codes. The performance was maintained for sentiment analysis and machine translation with 94% and 98% respective compression

rates. [385] propose a product quantization structured embedding that reduces memory by 10-20 times the number of parameters, while maintaining performance. This involves slicing the embedding tensor into groups which are then quantized independently. [319] also consider binary code for neural machine translation. However, their approach still required binary codes to be used in conjunction with the softmax and showed a performance degradation when only using binary codes. Here, we show that, when given enough bits, the model is competitive against the full softmax, and in some cases outperforming. Moreover, we introduce novel ways to mitigate EB in these models, and Latent Variable based models alike.

### 4.3 METHODOLOGY

In this section, we introduce the construction of the codebook that contains the error codes for each token and then describe how the codes can be semantically ordered to improve the approximation of the full posterior. From this, we then describe how ECOC can be used in conjunction with *mixture sampling* to mix codes of predicted and ground truth binary codes to mitigate exposure bias.

#### 4.3.1 Codebook Construction

A challenging aspect of assigning codewords is ordering the codes so that even if incorrect predictions are made, that the codeword is at least semantically closer to that of the codewords that are less related, while ensuring good separation between codes. Additionally, we have to consider the amount of error-checking bits to use. In theory,  $\log_2(k)/k$  is sufficient to account for all  $k$  classes. However, this alone can lead a degradation in performance. Hence, we also consider a large amount of error-checking bits. In this case, the error-checking bits can account for more mistakes given by other classes, which may be correlated. In contrast, using probability distributions naturally accounts for these correlations, as the mass needs to shift relative to the activation of each output. This is particularly important for LM and text generation because of the high-dimensionality of the output. The most naive way to create the codebook is to assign binary codes to each word in random order. However, it is preferable to assign similar codes to  $w \in \mathcal{V}$  that are semantically similar while maximizing the Hamming distance between codes where leftover error codes separate class codes.

##### 4.3.1.1 Codebook Arrangement

A fundamental challenge in creating the codebook  $\mathcal{C}$  is in how error-codes are distributed between codes that maximize the separability between codewords that are more likely to be interchangeably and

incorrectly predicted. This is related to the second challenge of choosing the dimensionality of  $C$ . The latter is dependent on the size of the corpus, and in some cases might only require  $|\log_2(\mathcal{V})| \leq d \leq |\mathcal{V}|$  bits to represent all classes with leftover error-checking bits. These two decisions correspond to a tradeoff between computational complexity and accuracy of our neural language model, akin to tree expressivity in the Hierarchical Softmax to using the Full Softmax. Below we describe a semantically motivated method to achieve well-separated codewords, followed by a guide on how to choose codebook dimensionality  $d_C$ .

**EMBEDDING SIMILARITY-BASED CODEBOOKS** Previous work on ECOC has focused on theories as to why randomly generated codes lead to good row and column separation [35]. However, this assumes that class labels are conditionally independent and therefore it does not apply well for sequence modelling where the output space is loosely structured. To address this, we propose to reorder  $C \in \mathcal{C}$  such that Hamming distance between any two codewords is proportional to the embedding similarity. Moreover, separating codewords by semantic similarity can be achieved by placing the amount of error-checking bits proportional to rank ordered similarity for a chosen query word embedding. A codebook ordered by pretrained word embedding similarities for  $w_*$  is denoted as  $\mathcal{C}_{\Lambda_{w_*}}$ . The similarity scores between embeddings is given as  $\mathcal{F}(\mathcal{M}_*, M_i) \forall i$  is used to reorder  $M \rightarrow \mathcal{M}'$ . In our experiments we use pretrained GoogleNews skipgram embeddings<sup>1</sup>. Good separation is achieved when codes are separated proportional to the cosine similarity between embeddings of the most frequent word  $w_* \in \mathcal{V}$  and the remaining words  $w'$ . Words with high similarity have corresponding codes that are closer in Hamming distance  $H(\cdot, \cdot)$  in  $\mathcal{C}$ .

This ensures that even when codes are correlated, that incorrect latent predictions are at least more likely to correspond to semantically related words. We are not guaranteed that codes close in Hamming distance are closer in a semantic sense in the random case. Therefore, we can instead consider computing  $\mathcal{M}'$  using ordered similarities of word embedding similarities where the function  $\mathcal{F}(\cdot, \cdot)$  computes the cosine similarity for any two words. Concretely, for  $k$  redundant codewords  $C_k$ , we require an assignment that leads to a strongly separated  $\mathcal{C}$ . For a function  $\delta(\cdot, \cdot)$  that assigns  $C_i^k$  error-checking codewords to the  $i^{th}$  class codeword and  $\delta(C_*, C_i) \propto \mathcal{F}(\mathcal{M}_*, \mathcal{M}_i) \forall k$ .

In practice  $\delta(\cdot, \cdot)$  normalizes the resulting embedding similarities  $S = \mathcal{F}(\mathcal{M}_*, M)$  using a normalization function  $\text{cumsum}(\phi(S) \times |\mathcal{C}|)$  to assign the intervals between adjacent codeword spans. This ensures greater distance between words that are more similar to  $w_*$ , and less error-checking codewords to rare words that have distant neighbouring words in the embedding space.

<sup>1</sup> available here: <https://code.google.com/archive/p/word2vec/>

### 4.3.2 Latent Variable Mixture Sampling

To mitigate EB for latent variable-based sequence modelling we propose a sampling strategy that interpolates between predicted and target codewords. We refer to this as Latent Variable Mixture Sampling (LVMS) and its application to ECOC as *Codeword Mixture Sampling* (CMS).

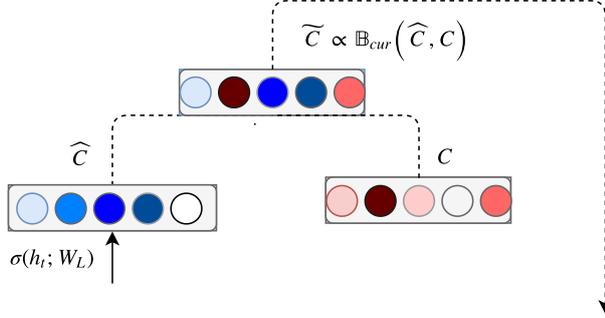


Figure 4.1: Curriculum Mixture Sampling

**CURRICULUM-BASED LATENT VARIABLE MIXTURE SAMPLING**  
 In Curriculum-Based Latent Variable Mixture Sampling (CLVMS), the mixture probability is  $p_c = 0 \forall c \in C$  at epoch  $\epsilon = 0$  and throughout training the probability monotonically increases  $p_c = \delta_c \forall c \in C$  where  $\delta_c$  is the threshold for the  $c$  th bit after  $\epsilon$  epochs. A Bernoulli sample  $\tilde{C} = \mathbb{B}(\hat{C}_c, C_c) \forall c \in [0, C]$  is carried out for  $t \in T$  in each minibatch. The probabilities per dimension  $p_c$  are independent of keeping a prediction  $\hat{y}_{t-1,c}$  instead of the  $c$  th bit in the target codeword  $y_{(t-1,c)}$  at timestep  $t-1$ . The reason for having individual mixture probabilities per bits is because when we consider a default order in  $C$ , this results in tokens being assigned codewords ranked by frequency. Therefore, the leftmost bit predictions are more significant than bit errors near the beginning (e.g  $2^0 = 1$  only 1 bit difference). We report results when using a sigmoidal schedule as shown in Equation 4.2 where  $\tau_{max}$  represents the temperature at the last epoch and  $\delta$  is a scaling factor controlling the slope.

$$[\hat{y}_{t-1}, y_{t-1}] \sim \frac{\tau_{max}}{1 + \exp(-\epsilon/\delta)}, \forall \epsilon \in [-N/2, N/2] \quad (4.2)$$

This is different to scheduled sampling since we are not just choosing between the ground truth and prediction, instead we are mixing factors of the predicted factored distribution and the target factored distribution that represents the posterior, as illustrated in Figure 4.1. Here, the color strength illustrates the activation between  $[0, 1]$ . Figure 4.1 also demonstrates how CMS is used in HS (without the additional error checks).

**LATENT SOFT-MIXTURE SAMPLING** In standard CMS, we pass token index  $w_t$  which is converted to an input embedding  $e_w$  based on the most probable bits predictions at the last time step  $\arg \max_{\theta} p(y_{t-1}|x_{t-1}; \theta)$ . We can instead replace the argmax operator with a soft argmax that uses a weighted average of embeddings  $e \in E$  where weights are assigned from the previous predicted output via the softmax normalization  $\phi(x_{t-1}, \tau)$ , where  $\tau$  controls the kurtosis of the probability distribution ( $\tau \rightarrow 0$  tends to argmax) in Equation 4.3.

$$x_t = \sum_{w \in \mathcal{V}} e_w \left( \frac{\exp(h_w^T \theta / \tau)}{\sum_{w \in \mathcal{V}} \exp(h_w^T \theta / \tau)} \right) \quad (4.3)$$

In the ECOC-NSP, we consider binary codewords and therefore choose the top  $k$  least probable bits to flip according to the curriculum schedule. Hence, this results in  $k$  codewords where each  $\hat{C}$  has at least hamming distance  $H(\hat{C}, C) = 1$  ( $2^0$ ). Concretely, this is a soft interpolation between past targets and a weighted sum of the  $k$  most probable codewords  $\hat{C}_K = \arg \max_k \left( \sigma(h_w^T W) \right)$  such that  $x_t = \mathbb{B}_K \left( C, \sum_k \phi(\hat{C}_k) \right)$  where  $B_K$  samples one or the other for each  $k$ th dimension of  $C$ .

#### 4.3.3 Differentiable Latent Variable Sampling

The previous curriculum strategies disregard where the errors originate from. Instead, they interpolate between model predictions of latent variables  $\hat{Y}$  and targets  $Y$  in a way that does not distinguish between cascading errors and localized errors. This means that it only recorrects errors after they are made instead of directly correcting for the origin of the errors. [269] showed that such operations can be approximated by using a continuous relaxation using the reparameterization trick, also known as the *Concrete Distribution*. By applying such relaxation it allows us to sample from the distribution across codes while allowing for a fully differentiable objective, similar to recent work [130]. We extend this to mixture sampling by replacing the argmax operation with the Concrete distribution to allow the gradients to be adjusted at points where prior predictions changed value throughout training. This not only identifies at which time-step the error occurs, but what latent variables (i.e. output codes) had the most influence in generating the error. This is partially motivated by the finding that in the latent variable formulation of simple logistic regression models, the latent variable errors form a Gumbel distribution. Hence, we sample latent codes inversely proportional to the errors from a Gumbel distribution.

**Gumbel-Softmax** Similarly, instead of passing the most likely predicted word  $\hat{y}_{t-1}^{w*}$ , we can instead sample from  $\hat{y}_{t-1} \sim \phi(h_{t-1}, w)$  and then pass this index as  $\hat{x}_t$ . This is an alternative to always acting greedily and allow the model to seek other likely actions. However, to compute

derivatives through samples from the softmax, we need avoid discontinuities, such as the argmax operation. The Gumbel-Softmax [183, 269] allows us to sample and differentiate through the softmax by providing a continuous relaxation that results in probabilities instead of a step function (i.e. argmax). As shown in Equation 4.4, for each component-wise Gumbel noise  $k \in [1.., n]$  for latent variable given by  $h^T \theta$ , we find  $k$  that maximizes  $\log \alpha_k - \log(-\log U_k)$  and then set  $D_k = 1$  and  $D_{\neg k} = 0$ , where  $U_k \sim \text{Uniform}(0, 1)$  and  $\alpha_k$  is drawn from a discrete distribution  $D \sim \text{Discrete}(\alpha)$ .

$$\hat{p}(y_t|x_t; \theta) = \frac{\exp((\log \alpha_k + G_k)/\tau)}{\sum_{i=1}^n \exp((\log \alpha_i + G_i)/\tau)} \quad (4.4)$$

For ECOC, we instead consider Bernoulli random variables which for the Concrete distribution can be expressed by means of two arbitrary Gumbel distributions  $G_1$  and  $G_2$ . The difference between  $G_1$  and  $G_2$  follows a Logistic distribution and so  $G_1 - G_2 \sim \text{Logistic}$  and is sampled as  $G_1 - G_2 \equiv \log U - \log(1 - U)$ . Hence, if  $\alpha = \alpha_1/\alpha_2$ , then  $P(D_1 = 1) = P(G_1 + \log \alpha_1 > G_2 + \log \alpha_2) = P(\log U - \log(1 - U) + \log \alpha > 0)$ . For a step function  $\mathcal{H}$ ,  $D_1 \equiv \mathcal{H}(\log \alpha + \log U - \log(1 - U))$ , corresponding to the Gumbel Max-Trick [183].

Sampling a Binary Concrete random variable involves sampling  $Z$ , sample  $L \sim \text{Logistic}$  and set  $Z$  as shown in Equation 4.5, where  $\alpha, \tau \in (0, \infty)$  and  $Z \in (0, 1)$ . This Binary Concrete distribution is henceforth denoted as  $\text{BinConcrete}(\cdot, \cdot)$  with location  $\alpha$  and temperature  $\tau$ . In the forward pass the probability  $Z$  is used to compute the approximate posterior, unlike the one-hot vectors used in straight-through estimation.

$$Z \equiv \frac{1}{1 + \exp\left(-(\log \alpha + L)/\tau\right)} \quad (4.5)$$

This is used for ECOC and other latent variable-based models, such as Hierarchical Sampling, to propagate through past decisions and make corrective updates that backpropagate to where errors originated from along the sequence. Hence, we also carry out experiments with  $\text{BinConcrete}$  (Equation 4.5) and Gumbel-Softmax (Equation 4.4) for HS and ECOC respectively. The temperature  $\tau$  can be kept static, annealed according to a schedule or learned during training, in the latter case this is equivalent to entropy regularization [131] that controls the kurtosis of the distribution. In this work, we consider using an annealed  $\tau$ , similar to Equation 4.2 where  $\tau \rightarrow 2.5$  and starts with  $\tau = 0.01$ . This is done to allow the model to avoid large gradient variance in updates early on. In the context of using the Gumbel-Softmax in LVMS, this allows the model to become robust to non-greedy actions gradually throughout training, we would expect such exploration to improve generalization proportional to the vocabulary size.

## 4.4 EXPERIMENTAL SETUP

Experiments are carried out for a 2-hidden layer Long-Short Term Memory (LSTM) model with embedding size  $|e| = 400$ , Backpropagation Through Time (BPTT) length 35 and variational dropout [118] with rate  $p_d=0.2$  for input, hidden and output layers. The ECOC-NSP model is trained using Binary Cross Entropy loss as shown in Equation 4.6, where  $k$  is a group error-checking codewords corresponding to a codeword  $C$ . The gradients can then expressed as  $\frac{\delta \mathcal{L}}{\delta \theta} = (y - \sigma(h^T \theta)) \cdot h^T$ .

$$\mathcal{L}_\theta = \max_k \prod_c^C \left[ y_c \log \left( \sigma_c(h^T \theta) \right) + (1 - y_c) \log \left( 1 - \sigma_c(h^T \theta) \right) \right] \quad (4.6)$$

**Baselines for ECOC-NSP** The first set of experiments compare the most related baselines ( Sample-Softmax [33, 34], Hierarchical Softmax (HS), AS [132] and NCE [290]) to our ECOC approach. For HS, we use a 2-hidden layer tree with a branching factor (number of classes) of  $\sqrt{|\mathcal{V}|}$  by default. For AS, we split the output into 4 groups via the unigram distribution (in percentages of total words 5%-15%-30%-100%). For NCE, we set the noise ratio to be 0.1 for PTB and 0.2 for WikiText-2 and WikiText-103. Training is carried out until near convergence ( $\epsilon \approx 40$ ), the randomly initialized HS and Sampled Softmax of which take longer ( $\epsilon \in [55-80]$ ). Table 4.1 reports the results for  $\log_2 |\mathcal{V}|^2$  number of samples in the case of Rand/Uni-Sample-SM. For Rand/Unigram Hierarchical SM, we use a 2-hidden layer tree with 10 classes per child node.

**Baselines for ECOC-NSP Mixture Sampling** To test Latent Variable Mixture Sampling (LVMS), we use it in HS and ECOC (two closely related latent NSP methods) and compare the performance of both. We also compare the performance of LVMS against the most related sampling-based technique for mitigating EB, scheduled sampling (SS) [31]. For SS used with cross-entropy loss (SS-CE), we also consider using a baseline of the soft-argmax (Soft-SS-CE) where a weighted average embedding is generated proportional to the predicted probability distribution.

**Evaluation Details** To compute perplexities for ECOC-NSP, the codewords are viewed as approximating the posterior via a factorial distribution [273] (a product of marginal probabilities) where each bit is independent of one another ( $c_i \perp c_j \forall i, j$ ). To compute the binary cross-entropy loss at training time, we choose the most probable  $k_i$  error checks corresponding to a tokens codeword  $C_{k_i}$ , as shown in Equation 4.7

At test time, if the predicted codeword  $\hat{C}$  falls within the  $k$  error-checking bits of codeword  $C$ , it is deemed a correct prediction and assigned the highest probability of all  $k$  predictions. Note that we only convert the ECOC predictions to perplexities to be comparable against baselines (if codes are easily decoded via some semantic distance or by Hamming distance, Hamming Distance or Mean Reciprocal Rank could also be used).

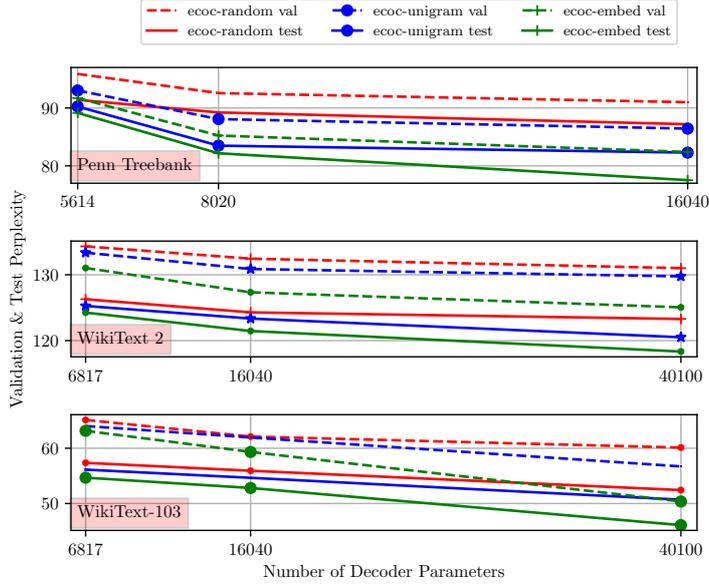


Figure 4.2: ECOC-NSP Perplexity vs. Decoder Parameters (corresponding to 14/20/40 codeword bits for Penn-TreeBank and 17/40/100 codeword bits for WikiText-2/103)

#### 4.5 RESULTS

**ERROR-CORRECTING OUTPUT CODED NSP** We first compare our proposed ECOC-NSP to aforementioned methods that approximate softmax normalization, using binary trees and latent codes that are ordered according to unigram frequency (Uni-Hierarchical-SM and Uni-ECOC). This is also the same ordering we use to compare our proposed CMS-ECOC sampling method to scheduled sampling [31] in standard cross-entropy training with softmax normalization.

$$p_{\hat{C}} = \max_{k_i} \prod_{c_{k_i}}^{C_{k_i}} \left( \phi(x_t, h_t) \right), \quad i = [0, 1, \dots, k] \quad (4.7)$$

Although, these are not directly comparable, since ECOC-NSP introduces a whole new paradigm, we use the common evaluation measures of Hamming distance and accuracy to have some kind of baseline with which we can compare our proposed method to.

Figure 4.2 shows how the reduction in perplexity as the number of ECOC-LSTM decoder parameters increase as more bits are added to the codeword. For PTB, large perplexity reductions are made between 14-100 codebits, while between 100-1000 codebits there is a gradual decrease. In contrast, we see that there is more gained from increasing codeword size for WikiText-2 and WikiText-103 (which preserve the words that fall within the long-tail of the unigram distribution). We find the discrepancy in performance between randomly assigned codebooks and ordered

Model	PTB		WikiText-2		WikiText-103	
	Val.	Test	Val.	Test	Val.	Test
Full SM	<b>86.19</b>	<b>79.24</b>	<b>124.01</b>	<b>119.30</b>	<b>56.72</b>	<b>49.35</b>
Rand-Sample-SM	92.14	81.82	136.47	129.29	68.95	59.34
Uni-Sample-SM	90.37	81.36	133.08	127.19	66.23	57.09
Rand-Hierarchical-SM	94.31	88.50	133.69	127.12	62.29	54.28
Uni-Hierarchical-SM	92.38	86.70	130.26	124.83	62.02	54.11
Adaptive-SM	91.38	85.29	118.89	120.92	60.27	52.63
NCE	96.79	89.30	131.20	126.82	61.11	54.52
Random-ECOC	91.00	87.19	131.01	123.29	56.12	52.43
Uni-ECOC	86.44	82.29	129.76	120.51	<b>52.71</b>	<b>48.37</b>
Embedding-ECOC	<b>84.40</b>	<b>77.53</b>	<b>125.06</b>	<b>120.34</b>	57.37	49.09

Table 4.1: Perplexities for Full Softmax (SM), Sampled SM (Sample-SM), Hierarchical-SM (HSM), Adaptive-SM, Noise Contrastive Estimation (NCE) & ECOC-NSP.

codebooks is more apparent for large compression ( $|C| < |\mathcal{V}|/10$ ). Intuitively, the general problem of well-separated codes is alleviated as more bits are added.

Equation 4.1 shows that overall ECOC with a rank ordered embedding similarity  $\mathcal{C}$  (Embedding-ECOC) almost performs as well as the full-softmax (8.02M parameters) while only using 1000 bits for PTB ( $|\mathcal{V}|/20$  and ) and 5K bits for WikiText-2 ( $|\mathcal{V}|/25$ ) and WikiText-103 ( $|\mathcal{V}|/30$ ). The HS-based models use a 2-hidden layer tree with 10 tokens per class, resulting in 4.4M parameters for PTB, 22.05M parameters for WikiText-2 (full softmax - 40.1M) and WikiText-103. Moreover, we find there is a consistent improvement in using Embedding-ECOC over using a random codebook (Random-ECOC) and a slight improvement over using a unigram ordered codebook (Uni-ECOC). Note that in both Embedding-ECOC and Uni-ECOC, the number of error-checking bits are assigned inversely proportional to the rank position when ordering embedding similarities (as discussed in Equation 4.3.1.1) and unigram frequency respectively. We also found that too many bits e.g  $|C| = |\mathcal{V}|$  takes much longer ( $\epsilon \in [20-30]$  more for PTB) to converge with negligible perplexity reductions. Hence, the main advantage of ECOC-NLVMS is the large compression rate while maintaining performance (e.g PTB with  $|C| = 40$ , there is less than 2 perplexity points compared to the full softmax).

LATENT VARIABLE MIXTURE SAMPLING TEXT GENERATION RESULTS Figure 4.3 shows how validation perplexity on WikiText-2 changes throughout training an LSTM as  $\tau$  begins to tend to  $\tau_{max} = 2.5$ ,  $\tau_{max} = 10$  and the case where  $\tau_{const} = 1$  is kept constant. We see that too much exploration ( $\tau_{max} = 10$ ) leads to an increase in perplexity, as

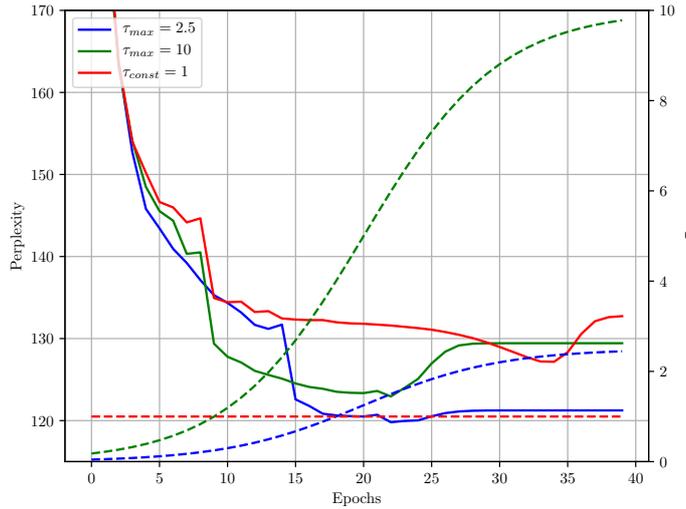


Figure 4.3: WikiText-2 Validation Perplexity When Varying  $\tau$  (corresponding dashed lines) in CLVMS-ECOC (LSTM)

$\tau > 5$ , the validation perplexity begins to rise. In contrast, we find a slow monotonic increase to  $\tau_{\max} = 2.5$  leads to a steady increase, at which  $\tau = 2$  (epoch 24) the model has almost converged.

Table 4.2 shows all results of LVMS when used in HS and ECOC-based NSP models for the MSCOCO image captioning dataset [244] using the Karpathy validation and test splits, with a beam search of width 5 at test time. We use  $|c| = 200$  to account for vocabulary size  $|V| = 10^3$ , leaving  $|c| - \log_2(|V|) = 186$  error-check bits leftover  $\forall C \in \mathcal{C}$ . The HS uses the Categorical Concrete distribution for DLVMS-HS and Binary Concrete Distribution for DCMS-ECOC. In our experiments we found  $\tau = 2.0$  to be the upper threshold from an initial grid search of  $\tau \in \{0.2, 0.5, 1, 2, 5, 10\}$  for both DLVMS-HS and DCMS-ECOC, where  $\tau < 2$  corresponds to little exploration and  $\tau > 2$  results in too much exploration, particularly early on when the model is performing larger gradient updates. CLVMS-Hierarchical-SM and CLVMS-ECOC both monotonically increases  $\tau$  according to Equation 4.2.

Both HS and ECOC use Embedding ordered decoder matrix (we omit the -Embedding extension). We baseline this against both SS and the soft-argmax version of SS, the most related sample-based supervised learning approach to LVMS. Furthermore, we report results on CLVMS-ECOC (Curriculum-LVMS ECOC) which mixes between true targets and codewords predictions according to the schedule in Equation 4.2 and a differentiable extension of LVMS via samples from the Gumbel-Softmax (DCMS-ECOC). For both DCMS-ECOC and DLVMS-Hierarchical-SM we sample from each softmax defined along the path to the target code at training time. We find that using a curriculum in CLVMS-ECOC with a semantically ordered codebook outperforms the full softmax that uses scheduled sampling (SS-SM) and its weighted-

	<b>B1</b>	<b>B2</b>	<b>B3</b>	<b>B4</b>	<b>R-L</b>	<b>MET</b>
Full-SM	71.09	51.33	32.85	24.67	50.28	52.70
SS-SM	73.23	52.81	33.37	26.11	52.60	54.51
Soft-SS-SM	<b>73.54</b>	<b>53.01</b>	<b>33.26</b>	<b>27.13</b>	<b>54.49</b>	<b>54.83</b>
SS-Adaptive-SM	70.45	50.22	31.38	23.59	51.88	51.83
SS-Hierarchical-SM	67.89	48.42	30.37	22.91	49.39	50.48
CLVMS-Hierarchical-SM	69.70	49.52	31.91	24.19	51.35	51.20
DLVMS-Hierarchical-SM	71.04	50.61	32.26	24.72	52.83	52.36
SS-ECOC	72.02	52.03	32.57	25.42	51.39	53.51
Soft-SS-ECOC	72.78	53.29	33.15	25.93	52.07	54.22
CLVMS-ECOC	<b>74.70</b>	<b>53.09</b>	<b>34.28</b>	<b>27.05</b>	<b>53.67</b>	<b>55.62</b>
DLVMS-ECOC	<b>74.92</b>	<b>53.56</b>	<b>34.70</b>	<b>27.81</b>	<b>54.02</b>	<b>55.85</b>

Table 4.2: MSCOCO Test Results on BLEU (B), ROUGE-L (R-L) & METEOR (MET) Eval. Metrics

variant (Soft-SS-SM). Moreover, DLVMS-ECOC further improves over CLVMS-ECOC on MSCOCO. LVMS makes a consistent improvement over SS. This suggests that LVMS is an effective alternative for latent variable based NSPs in particular (mixture sampling is ill-suited to one-hot targets as they are extremely sparse).

#### 4.6 SUMMARY

The work presented in this chapter proposed an error-correcting neural language model and a novel Latent Variable Mixture Sampling method for latent variable models. We find that performance is maintained compared to using the full conditional and related approximate methods, given a sufficient codeword size to account for correlations among classes. This corresponds to 40 bits for PTB and 100 bits for WikiText-2 and WikiText-103. Furthermore, performance is improved when rank ordering the codebook via embedding similarity where the query is the embedding of the most frequent word.

Lastly, Latent Variable Mixture Sampling was introduced to mitigate exposure bias. This can be easily integrated into training latent variable-based language models, such as the ECOC-based language model. This method outperforms the well-known scheduled sampling method with a full softmax, hierarchical softmax and an adaptive softmax on an image captioning task, with less decoder parameters than the full softmax with only 200 bits, 2% of the original number of output dimensions.

# 5

---

## MODEL COMPRESSION VIA META-EMBEDDING

---

The previous chapter focused on compression of the *output* layer of **DNNs**. In contrast, this chapter discusses how to compression the *input*, specifically an ensemble set of pretrained word embeddings. This approach involves reconstructing concatenated pretrained word embeddings with an autoencoder whereby the hidden representation output is shared and also used as input to a downstream task classifier. Hence, there is two losses for 1) the meta-embedding reconstruction and 2) the main task classification. This supervised meta-embedding approach improves over self-supervised meta-embedding for specific tasks. We now begin with the introduction.

### 5.1 INTRODUCTION

Word embeddings have been shown to benefit from ensembling several word embedding sources, often carried out using straightforward mathematical operations over the set of word vectors. More recently, self-supervised learning has been used to find a lower-dimensional representation, similar in size to the individual word embeddings within the ensemble. However, these methods do not use the available manually labeled datasets that are often used solely for the purpose of evaluation. We propose to reconstruct an ensemble of word embeddings as an auxiliary task that regularises a main task while both tasks share the learned meta-embedding layer. A set of experiments are carried out for intrinsic evaluation (6 word similarity datasets and 3 analogy datasets) and extrinsic evaluation (4 downstream tasks). For intrinsic task evaluation, supervision comes from various labeled word similarity datasets. The experimental results show that the performance is improved for all word similarity datasets when compared to self-supervised learning methods with a mean increase of 11.33 in Spearman correlation. Specifically, the proposed method shows the best performance in 4 out of 6 of word similarity datasets when using a cosine reconstruction loss and Brier’s word similarity loss. Moreover, improvements are also made when performing word meta-embedding reconstruction in sequence tagging and sentence meta-embedding for sentence classification. Lastly, a Target Autoencoder (TAE) is also proposed to predict embeddings in

the ensemble set from the remaining embeddings. This has the benefit of filling in embeddings for which there is missing words in the pretrained word embedding vocabulary.

Distributed word representations have shown to improve performance in numerous natural language processing (NLP) tasks [194, 214, 251, 421]. Given that the performance on intrinsic (e.g word similarity, analogy) and extrinsic (e.g sentiment analysis, dependency parsing, machine translation etc.) supervised tasks is dependent on the model used for producing word embeddings (e.g `skipgram`, `cbow` [281]), it is clear that each model exploits different aspects of the semantic space. The goal in meta-embedding learning [22, 41, 66, 196, 468] is to learn a single, (possibly lower-dimensional) common embedding space by combining multiple, pre-trained *source* word embeddings, without re-training the sources or requiring access to the linguistic resources such as text corpora or dictionaries that were used to train the source embeddings. Meta-embedding learning methods aim to be computationally and resource-wise efficient by not retraining source embeddings.

Existing approaches to meta-embedding rely on self-supervised learning such as autoencoding [22] to find a lower-dimensional hidden representation of the set of source embedding as discussed in Section 2.4.5. This can be advantageous in cases where; (1) pre-training is expensive, (2) pre-trained embeddings are available but not the algorithm or the training data used, and (3) the available source embeddings vary in their dimensionalities. However, for problems that rely on representations that are better aligned with human judgment (e.g., *genuine* similarity [159]), word embeddings and word meta-embeddings struggle to perform well when only given co-occurrence statistics. How to best incorporate task-specific human judgements into the meta-embedding learning process remains a challenging and unsolved problem.

To overcome this challenge, we propose a semi-supervised multi-task learning (MTL) approach that combines the benefits of self-supervised learning for finding a lower-dimensional representation of the concatenated word meta-embeddings, while also learning to perform inference on word similarity used as an intrinsic task, and extrinsic downstream tasks such as sequence tagging using a siamese network that incorporates a shared representation from an autoencoder (AE). We consider meta-embedding reconstruction as an auxiliary task in contrast to the main classification task, hence the reference to multi-task learning. We evaluate our approach on held-out word similarity datasets and also include an evaluation on the transferability of the resultant word meta-embeddings on three analogy datasets. We find that performance is improved for *all* word similarity datasets with a mean increase of 11.33 points in the Spearman Correlation coefficient  $\rho_s$ , when compared to self-supervised learning methods. Below we summarize the main aspects of our work.

**ANGULAR COST FUNCTIONS** In meta-embedding learning we use source embeddings trained on different resources, in which case it is important to keep the semantic orientation of words by preserving the angle between word embeddings, not only the length. Cosine similarity, a popularly used measure for computing the semantic relatedness among words, ignores the length related information. We also note the relationship between KL-divergence and cosine similarity in that both methods perform a normalization. Hence, we compare Mean Squared Error (MSE) and Mean Absolute Error (MAE), against KL-divergence and Squared Cosine similarity for the purpose of learning meta-embeddings and show that loss functions that account for vector orientation can outperform the former MSE and MAE objectives that only preserve length but not orientation.

**SUPERVISION FROM MANUAL ANNOTATIONS** Currently, word embeddings and word meta-embedding methods do not exploit the available manual annotations in the learning process such as word similarity ratings. In particular, word vectors often struggle to preserve true similarity, which in many cases is difficult to identify from statistical associations alone. Hill et al. [159] found word embeddings to struggle for word similarity in comparison to word association, particularly for abstract concepts. Our method addresses this by learning to reconstruct meta-embeddings while sharing the hidden layer to jointly predict on a main task (e.g word similarity, NER or Sentiment classification). In the case of word similarity, we argue that this explicit use of true similarity scores can greatly improve embeddings for tasks that rely on true similarity. This is reflected in our results for Simlex [158] and rare word [265] datasets as we find 29.63 and 27.05 point increases in  $\rho_s$  respectively.

**DEALING WITH OUT-OF-VOCABULARY WORDS** Word vectors suffer in performance for out-of-vocabulary words that are not seen during training. This is an issue on evaluation datasets that gauge performance on words that are morphologically complex, rare [266] or are highly abstract conceptually [159]. In fact, Luong et al. [266] have used sub-word vectors for such issues. Alternatively, Cao and Rei [53] have used a character-level composition from morphemes to word embeddings where morphemes that yield better predictive power for predicting context words are given larger weights, showing improvements over word-based embeddings for syntactic analogy answering. Their model incorporated morphological information into character-level embeddings which in turn, produced better representations for unseen words. Word meta-embeddings allow for much larger coverage by combining the ensemble set of pre-trained word embeddings, trained from different corpora. This approach also allows for sub-word level combinations between embeddings.

MOTIVATION FOR MULTI-TASK LEARNING: Using shared representations in multi-task learning has led to better generalization performance on each respective task in prior work [54], by (1) introducing relevant inter-dependent features across related tasks, (2) regularizing a model to generalize for multiple tasks, (3) using future tasks to interpolate to present tasks, (4) improving the model’s ability to learn general features from noisy signals, and (5) potentially exploiting the loose structure among the parent tasks that aid more specific downstream child sub-tasks (e.g tasks are designated based on the word relation type such as hyponymy, antonymy or synonymy).

### 5.1.1 *Multi-Task Learning Representations*

Ando and Zhang [8] learn representations for multiple tasks in a partially supervised and unsupervised way, which draws similarities to the work presented in this chapter. The challenge is characterized as (1) predicting labels for an auxiliary task from another task that is trained with full supervision, and (2) both tasks are in some way related. Both characteristics also hold for the work presented in this chapter with the subtle difference that we are using many related word similarity datasets with full supervision to predict the auxiliary task (i.e a held-out word similarity dataset for testing). Part-of speech (PoS) tags of the contextual words are used to predict the current word’s PoS tag in a self-supervised fashion, similar to masking used in word embedding learning [90, 281]. Specifically, some features that are to be predicted for text categorization are masked out to create auxiliary prediction tasks. Ando and Zhang [8] used this method to obtain good performance on PoS tagging and Named Entity Recognition (NER) using a language model that predicts a target word given its context words. Similarly, Collobert et al. [67] proposed a unified neural network architecture for learning PoS tagging, NER, Chunking and Semantic Role Labeling all at once with parameter sharing using unlabeled training data.

Dong et al. [95] used MTL to improve the quality of machine translation to multiple target languages. They share the source language representations in the encoder-decoder sequence model considering the availability of the required parallel data. Their model showed higher BLEU scores over independent sequence-to-sequence language models when there is full and partial availability of parallel data, highlighting the importance of integrating related source language representations.

Liu et al. [253] used MTL for query classification using multiple binary classifiers, and web search ranking based on maximum likelihood with deep neural networks. Their MTL architecture consisted of three shared hidden layers that use character and word  $n$ -gram inputs, where the last layer is an independent task-specific layer for query classification and web search respectively. MTL showed large improvements over the

baseline support vector machines and neural networks that learn each task independently.

All of the aforementioned work focus on using MTL on high-level natural language tasks. This work is distinct in that we use meta-embedding reconstruction as a regularization technique, treating it as an auxiliary task to the main intrinsic or extrinsic task of interest.

## 5.2 METHODOLOGY

Before introducing the semi-supervised MTL approach to learning word meta-embeddings, we first outline the self-supervised learning baselines used in the comparisons. First, we include both the aforementioned 1TON/1TON<sup>+</sup> [468] and standard AEs [22] proposed in the prior work. CAEME concatenates the embedding set into a single vector and trains the AE to produce a lower-dimensional representation, while DAEME keeps the embedding vectors separate in the encoding.

## 5.3 AUTOENCODERS FOR META-EMBEDDING

We consider a dataset of  $n$  samples,  $D := \{(\mathbf{x}_i, \mathbf{y}_i)\}^n$ , where the input is represented as a word embedding input matrix  $\mathbf{x} \in \mathbb{R}^{|V| \times d}$  with a corresponding target vector  $\mathbf{y} \in \mathbb{R}^{|V|}$ . Here  $x_i$  is the embedding for a word  $w \in V$  where  $V$  denotes the vocabulary for  $D$ . Further  $V \subset \mathcal{V}$  is a subset vocabulary for a given  $D \in \mathcal{D}$  where  $\mathcal{D}$  are all datasets and unique set of words for  $\mathcal{D}$  is  $\mathcal{V}$ .

The matrix for all  $D \in \mathcal{D}$  and  $N$  embeddings in the ensemble set of word embeddings  $X : x_1, x_2, \dots, x_N$  can be expressed as a matrix  $\mathbf{X} \in \mathbb{R}^{n \times k}$  which can be split into training matrix  $\mathbf{X}_{tr} \in \mathbb{R}^{n_{tr} \times k}$  and test matrix  $\mathbf{X}_{ts} \in \mathbb{R}^{n_{ts} \times k}$ . We define  $k = Nd$ ,  $n_{tr} = |\mathcal{V}| - |V|$  to be the number of training examples (one  $D$  is left out for training),  $n_{ts} = |V|$  number of test examples, both obtained by row-wise concatenation of pretrained embeddings, as shown in Equation 5.1 and for simplicity, refer to  $\mathbf{X}_{tr}$  as  $\mathbf{X}$  herein. In standard meta-embedding training, we obtain a hidden representation  $\mathbf{Z} \in \mathbb{R}^{n_{tr} \times k}$  from  $\mathbf{X}$  where  $k \ll Nd$  using an AE  $f_\theta$ .

$$\mathbf{Z}(w) = f_\theta(\mathbf{X}(w)), \quad \mathbf{X} := \bigoplus_{i=1}^N \mathbf{x}_i \quad (5.1)$$

In Section 5.4 we will discuss how  $\mathbf{Z}$  is learned as an auxiliary task to the main supervised task which also shares  $\mathbf{Z}$ . We now consider reconstruction loss functions.

### 5.3.0.1 Meta-Embedding Loss Functions

We compared training AEMEs with various loss functions against other meta-embedding approaches mentioned in related work such as

1TON [468] and autoencoded meta-embeddings [22]. This includes the MAE loss  $\sum_{i=1}^N |\mathbf{X}_i - \hat{\mathbf{X}}_i|$ , MSE loss  $\sum_{i=1}^N (\mathbf{X}_i - \hat{\mathbf{X}}_i)^2$  and the KL divergence. For minimizing the KL divergence, the AE output distributions for each sample in  $\hat{\mathbf{X}}$  are normalized to form  $\hat{p}(\mathbf{X})$  and the corresponding meta-embedding target distribution  $p(\mathbf{X})$  using the `softmax` function. The KL is then expressed as Equation 5.2 which corresponds to the reconstruction loss  $\mathcal{L}_r$  shown in Equation 5.3. Here,  $M$  refers to the mini-batch size used for training.

$$D_{\text{KL}}\left(p(\mathbf{X})||p(\hat{\mathbf{X}})\right) = \sum_{i=1}^M p(\mathbf{X}_i) \log \frac{p(\hat{\mathbf{X}}_i)}{p(\mathbf{X}_i)} \quad (5.2)$$

$$\mathcal{L}_r(\hat{\mathbf{X}}, \mathbf{X}) = \frac{1}{M} \sum_{i=1}^M p(\mathbf{X}_i) \left( \log(p(\mathbf{X}_i)) - \log(\hat{p}(\hat{\mathbf{X}}_i)) \right) \quad (5.3)$$

Since tanh activations are used and input vectors are  $\ell_2$  normalized we propose a Squared Cosine Proximity (SCP) loss, shown in Equation 5.4. This forces the optimization to tune weights such that the rotational difference between the embedding spaces is minimized, thus preserving semantic information in the reconstruction. In the context of its utility for the TAE, we also want to minimize the angular difference between corresponding vectors in different vector spaces. Unlike KL-divergence, the SCP loss is a proper distance metric since it is symmetric and satisfies the triangular inequality.

$$\mathcal{L}_r(\hat{\mathbf{X}}, \mathbf{X}) = \sum_{i=1}^M \left( 1 - \frac{\sum_{j=1}^m \hat{\mathbf{X}}_{ij} \cdot \mathbf{X}_{ij}}{\sqrt{\sum_{j=1}^m \hat{\mathbf{X}}_{ij}^2} \sqrt{\sum_{j=1}^m \mathbf{X}_{ij}^2}} \right)^2 \quad (5.4)$$

### 5.3.1 Model Configurations

The AEME is a 1-hidden layer AE with a hidden layer dimension  $d_h = 200$ . This is consistent for all tested loss functions, making for a fair performance comparison between the proposed SCP loss and KL divergence loss against MSE and MAE. We initialize all weights with a normal distribution of mean  $\mu = 0$  and standard deviation  $\sigma = 1$ . The dropout rate is set to  $p = 0.2$  for all datasets. The model takes  $|\mathcal{V}| = 4819$  unique vocabulary terms pertaining to all tested word association and word similarity datasets and performs Stochastic Gradient Descent (SGD) with a mini-batch size of 32 trained for 50 epochs for each dataset with an early stopping criteria. The hidden dimension size, batch size and number of epochs were chosen based on a small grid search.

### 5.3.2 Target Autoencoder Meta-Embedding

We propose the Target AE (TAE), an AE variant that learns to predict a target embedding  $\mathbf{X}_{tr}^t \in \mathbb{R}^{n_{tr} \times d}$ , in the meta-embedding set, from the remaining source embeddings  $\mathbf{X}^s \in \mathbb{R}^{n_{tr} \times d(N-1)}$ . The AE  $\mathbf{f}_{\Theta_i} : \mathbf{X}_{tr}^{(s,i)} \rightarrow \mathbf{X}_{tr}^t \quad \forall i \in N$  permutations within the embedding set in a leave-one-out setting, where each  $i$ -th model is trained for each permutation. We then denote resulting meta-embedding as  $\bar{\mathbf{Z}}_i$  and  $\mathbf{Z} := \bar{\mathbf{Z}}$  where  $k = d$  in our experiments for the TAE embedding. This embedding represents different combinations of mappings from one vector space to another. This is motivated by [54] who points out that treating inputs as auxiliary output tasks can be beneficial. This has also found in the success of large-scale language modelling where predicting a percentage (e.g 15%) of masked tokens from unmasked tokens has led to better representations as measured on supervised tasks [90].

In contrast, the TAE is similar to that of CAEME only the label is a single embedding from the embedding set and the input are remaining embeddings from the set. After training a TAE, the hidden layer encoding is concatenated with the original target vector. The Mean Target AutoEncoder (MTE) instead takes an average between different projections.

## 5.4 META-EMBEDDING FOR SUPERVISED TASK REGULARIZATION

This section details the proposed supervised meta-embedding approach that uses multi-task learning. The below subsection details the setup for intrinsic tasks (word similarity and analogies) and then followed by extrinsic tasks (sentence classification and structured prediction).

### 5.4.1 Multi-Task Learning for Intrinsic Tasks

We begin with intrinsic tasks, specifically word similarity datasets to evaluate how supervised meta-embeddings compare to pretrained word embeddings, self-supervised meta-embeddings and the proposed method with varying reconstruction losses.

**WORD SIMILARITY** The first instance of using meta-embeddings as an auxiliary reconstruction task is for learning word associations. Since, we are combining self-supervised learning (reconstruction of the ensemble set) and supervised learning (word association scores) simultaneously, we view this approach as semi-supervised learning. The shared representation that is used during reconstruction is also shared as input for the main task.

The word similarity scores  $\mathbf{y} \in [0, 1]$  are normalized as different datasets are within different ranges. The resulting normalized  $\mathbf{y}$  are

considered as soft probabilistic targets. We also considered converting  $\mathbf{y}$  to binary classes using a threshold from the mean  $\bar{y}$ . However, as illustrated in Figure 5.1, the quartiles of the distribution over the annotation scores are not symmetric around the median, with the exception of MEN and Simlex. Moreover, factors such as annotation guidelines, part of speech (PoS) distribution and the concreteness of word pairs are different for each dataset. Such factors partially explain why the output distributions  $\forall y \in \mathcal{Y}$  ( $\mathcal{Y}$  corresponds to all outputs in  $\mathcal{D}$ ) vary as seen in Figure 5.1.

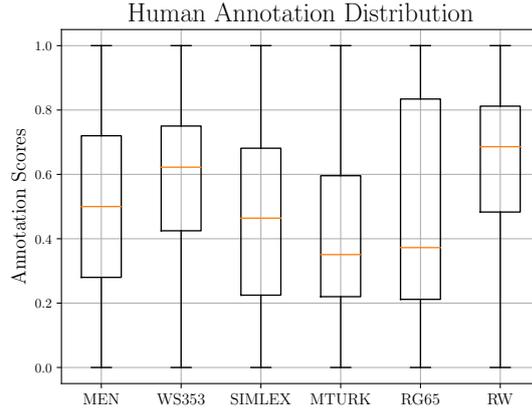


Figure 5.1: Word Association Annotation Distribution

We train on all but one word similarity test dataset  $D_{ts}$  using the AE to produce meta-embedding pairs  $h^1$  for the word pair vectors that are also used on the test dataset as it is the unsupervised (self-supervised) learning part of the network. This is illustrated in Figure 5.2, where red coloring indicates the hidden layer representations.

For training, the hidden layer dimension sizes are  $200 - 50 - 10$ , corresponding to  $h^{(1)}-h^{(2)}-h^{(3)}$  in Figure 5.2, which shows an example of *true* similar where ‘teacup’ and ‘teabag’ are distinctly different. For the layers that are only used on the main task ( $h^{(2)}$  and  $h^{(3)}$ ) we denote their parameters as  $\Theta$ . This notation is also used to refer intrinsic task-specific layers. Furthermore we define the parameters of the main tasks final layer as  $\omega \subset \Theta$ .

Note that  $h^{(1)}$  has dimensionality  $d = 200$ , the same size as the aforementioned self-supervised learning approach that does not use MTL (ie word similarity is not learned). Once MTL has converged over a set of epochs and hyperparameters are tuned, we compare the  $\rho_s$  of the shared hidden layer  $h^{(1)}$  outputs, as opposed to using  $\hat{y}$  produced in the siamese network that predicts word similarity directly. We test various distance measures for word similarity, including Manhattan, Euclidean and Cosine dissimilarity.

Since the data is [0-1] normalized the pairwise distance can be computed as  $\hat{y} = \exp(-d_\omega(\mathbf{h}_1^l, \mathbf{h}_2^l))$  and is kept in this range using the negative exponent. This corresponds to estimating the probability density of the output targets, where  $y$  are soft probabilistic targets.

The total loss  $\mathcal{L} = \mathcal{L}_r + \mathcal{L}_s$  is then the sum of the reconstruction loss  $\mathcal{L}_r$  shown in Equation 5.5 and  $\mathcal{L}_s$ , the cross-entropy (CE) loss between predictions made by the meta-embedding shared representation shown in Equation 5.6. In Equation 5.5,  $\lambda$  controls the amount of meta-embedding regularization during optimization.

$$\mathcal{L}_r(\mathbf{X}, \hat{\mathbf{X}}) = \frac{\lambda}{2M} \sum_{i=1}^M \sum_{j=1}^2 \left( \mathbf{X}_{ij} - \hat{\mathbf{X}}_{ij} \right)^2 \quad (5.5)$$

$$\mathcal{L}_s(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{M} \sum_{i=1}^M \mathbf{y}_i \log \hat{\mathbf{y}}_i + (1 - \mathbf{y}_i) \log (1 - \hat{\mathbf{y}}_i) \quad (5.6)$$

We argue that when annotators decide on word similarity given a word pair that they choose on how  $x_1$  relates to  $x_2$  only, and not vice-versa [422]. In other words, the relationship between two words is not strictly symmetric and the viewing order matters when humans are tasked with estimating word similarity. Therefore, when coming up with a similarity measure using this argument, we test an asymmetric similarity measure between  $(h_2^l, h_1^l)$  encodings. This simply involves replacing the denominator of the cosine similarity ( $\|x_1\|_2 \|x_2\|_2$ ) that is used as the distance function  $d_\omega(\cdot)$  to  $\|x_1\|_2 \cdot \|x_1\|_2$  before being passed to the negative exponent for the final probability output. Hence, the model is trained to learn how  $x_1$  is related to  $x_2$ .

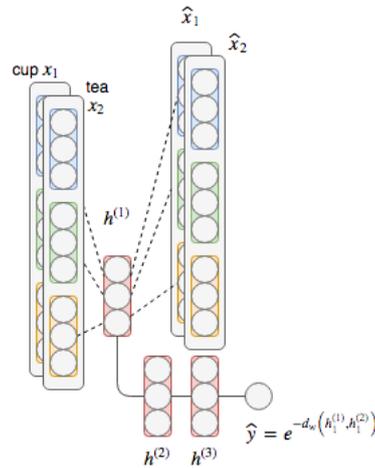


Figure 5.2: Multi-Task Meta-Embedding (an example of *true* similarity). An illustration of how the meta-embeddings of cup and tea are used as input to the proposed architecture for word similarity tasks, where the meta-embedding hidden layer is shared for reconstruction and word similarity prediction.

**MULTI-TASK LEARNING FOR ANALOGY** We also test our meta-embeddings as an auxiliary task for analogy, a task that plays a fundamental role in human cognition [122, 123]. Since not all algorithms used to train the pretrained embeddings are known to preserve analogies as a side effect, we might expect single embeddings for which this does hold to outperform meta-embeddings. We test if the word meta-embedding encodes analogical structure. Furthermore, we test if meta-embedding reconstruction improves performance.

For analogy answers, we focus on CosAdd [284] for measuring similarity between analogy pairs and ranking candidates accordingly. Hence, the meta-embedding scheme can be expressed as Equation 5.7, where  $(w_a, w_b)$  are the first analogy pair and  $(w_c, w_d)$  are the second, while  $Z(w)$  is the meta-embedding representation for a given  $w$ .

$$\begin{aligned} \text{CosAdd}(\mathbf{Z}(w_a), \mathbf{Z}(w_b), \mathbf{Z}(w_c), \mathbf{Z}(w_d)) = \\ \text{Cos}(\mathbf{Z}(w_b) - \mathbf{Z}(w_a) + \mathbf{Z}(w_c), \mathbf{Z}(w_d)) \end{aligned} \quad (5.7)$$

#### 5.4.2 Multi-Task Learning for Extrinsic Tasks

Finally, we test how such meta-embeddings perform in standard supervised learning problems in natural language tasks, more specifically, sequence prediction and text classification problems. This includes Named Entity Recognition (NER), Universal Dependency Part-of-Speech (UD-PoS) tagging

We learn to reconstruct the embeddings using the top performing autoencoding approach and use it to perform meta-embedding as an auxiliary task for log-likelihood training of one of the above sequence tagging problems. In this setting,  $Y$  is a sequence of length  $T$  with tokens  $y_1, y_2, \dots, y_T$ ,  $\Theta$  are the parameters of an RNN with encoded hidden state vector  $h_t = f(x_t, h_{t-1}; \Theta)$  and  $P(y_t|h_t; \Theta)$  is the conditional computed using a linear projection followed by normalization, in our case, using a `softmax` function.

$$\Theta^* = \arg \max_{\Theta} \sum_{t=1}^T \log P(y_t|x_t, h_{t-1}; \Theta) \quad (5.8)$$

The model uses tokens  $x_t$  as the input and the hidden state  $h_{t-1}$  to predict a tag  $\hat{y}_{t-1}$ . We find the optimal sequence of tags such that the likelihood of the predicted sequence of tags is maximised.

In the subsequent analysis, we learn to reconstruct the embeddings as in Equation 5.5 as an auxiliary task. However, instead of using two dense layers for predicting on the main task, as is the case for word similarity, we use the output of the shared representation as input to recurrent and convolutional layers in the aforementioned downstream tasks. We add a penalty  $\lambda \mathcal{L}_r$ , where the penalty coefficient  $\lambda \in \mathbb{R}$  is tuned based

on validation performance, effectively constraining optimization to also reconstruct the embedding set.

## 5.5 EXPERIMENTS

The following source word embeddings are considered in the embedding set as they are publicly available and widely used for natural language tasks: skipgram and CBOW [281], Glove [335], FastText [37], LexVec [369], Hellinger PCA (HPCA) [225] and Hierarchical Document Context (HDC) [401].

### 5.5.1 *Intrinsic Evaluation*

**WORD SIMILARITY RESULTS** The following word association and word similarity datasets are used throughout experimentation: Simlex [158], WordSim-353 [108], RG [365], MTurk (MechanicalTurk-771) [138], rare word (RW) [265] and MEN [47]. Table 5.1 shows the results, where (1) shows the single embedding performance, (2) results for standard meta-embedding approaches that either apply a single mathematical operation or use a linear projection as an encoding, (3) results using AE schemes by Bollegala and Bao [22] and (4) results of our proposed TAE embedding. Results in red shading indicate the best performing meta-embedding for all presented approaches (with the exception of skipgram on WS353), while black shading indicates the best performing meta-embedding for that cell block e.g Lexvec is shaded black on MTurk because it is the best word embedding within the “1. Embedding” block.

Best performing word meta-embeddings are held between CAEMEs that use the proposed Cosine-Embedding loss, while KL-divergence also performs well on Simlex and RW. Interestingly, both Simlex and RW are distinct in that Simlex is the only dataset providing scores on *true similarity* instead of free association, which has shown to be more difficult for word embeddings to account for [159], while RW provides morphologically complex words to find similarity between. This suggests KL-divergence is well suited for encoding word relations that are relatively rare and perhaps more abstract relations (e.g Simlex contains less concrete terms in the vocabulary compared to other datasets, such as WS353). Similarly, we find SCP loss to achieve best results on RG and MEN, both the smallest and largest datasets of the set.

Furthermore, the TAE variant has lead to competitive performance against other meta-embedding approaches, showing good results on WS353. However, overall, standard AE performs better than the TAE.

The AE that uses a squared cosine loss and a KL-divergence loss improves performance in the majority of the cases, reinforcing the argument that considering the angles explicitly through normalization (log-softmax for KL) is an important step in encoding large documents of varying length and semantics. Lastly, we have shown its use in the

1. Embeddings	Simlex	WS353	RG	MTurk	RW	MEN
Skipgram	44.19	77.17	76.08	68.15	49.70	75.85
FastText	38.03	75.33	79.98	67.93	47.90	76.36
GloVe	37.05	66.24	76.95	63.32	36.69	73.75
LexVec	41.93	64.79	76.45	71.15	48.94	80.92
HPCA	16.60	57.11	41.72	37.45	13.36	34.90
HDC	40.68	76.81	80.58	65.76	46.34	76.03
2. Standard Meta						
CONC	42.57	72.13	81.36	71.88	49.91	80.33
SVD	41.10	72.06	81.18	71.50	49.13	79.85
AV	40.63	70.50	80.05	70.51	49.28	78.31
1TON	41.30	70.19	80.20	71.52	50.80	80.39
1TON*	41.49	70.60	78.40	71.44	50.86	80.18
3. $\ell_2$ -AE						
Decoupled	42.56	70.62	82.81	71.16	50.79	80.33
Concatenated	43.10	71.69	84.52	71.88	50.78	81.18
$\ell_1$ -AE						
Decoupled	43.52	70.30	82.91	71.43	51.48	81.16
Concatenated	44.41	70.96	81.16	69.63	51.89	80.92
Cosine-AE						
Decoupled	43.13	71.96	84.23	70.88	50.20	81.02
Concatenated	44.85	72.44	85.41	70.63	50.74	81.94
KL-AE						
Decoupled	44.13	71.96	84.23	70.88	50.20	81.02
Concatenated	45.10	74.02	85.34	67.75	53.02	81.14
4. TAE +Y						
→ Skipram	42.43	75.33	80.11	66.51	44.77	78.98
→ FastText	41.69	72.65	80.51	67.64	47.41	77.48
→ Glove	41.75	76.65	82.40	68.92	48.83	78.27
→ LexVec	42.85	73.33	80.97	69.17	46.71	79.63
→ HPCA	40.03	69.65	70.43	61.31	36.38	73.10
→ HDC	42.43	74.08	80.11	66.51	44.76	77.93

Table 5.1: Meta-Embedding Unsupervised Results ( $\rho_s$ )

context of training word meta-embedding but cosine loss can also be used to minimize angular differences in standard word embedding training.

Table 5.1 shows the results for the self-supervised learning methods, where grey represents the best model for each section (1-4) and red represents the best model for all sections (same for proceeding tables). It is clearly difficult to obtain relatively good performance on Simlex and RW. The former was introduced to make a clear distinction between association and true similarity, hence the annotation scores reflect this difference, making it difficult for DSMs which solely rely on word associations. In contrast, we see in Table 5.3 there is a large improvement in  $\rho_s$  over these datasets using SS-MTL. In experimentation, we found performance with a 2-hidden layer network was similar to a single layer network. Given the sample size of  $|\mathcal{V}| = 4819$  for the word similarity,

we are not surprised that a relatively smaller sample size relies less on a deeper representation.

The first measure (e.g Cosine-) represents the reconstruction loss  $\mathcal{L}_r$  and second represents the word similarity loss  $\mathcal{L}_s$  (e.g Binary CE). A cosine  $\mathcal{L}_r$  and the Brier’s score  $\mathcal{L}_s^1$  are found to perform the best on average.

	MSR	Google	SemEval
Skipgram	73.13	72.89	22.64
FastText	64.19	73.82	24.77
GloVe	71.45	71.73	19.98
LexVec	74.03	67.28	21.49
Cosine-OLS	73.24	71.57	22.13
Cosine-NLL	71.23	68.39	20.16
Cosine-Brier’s	74.78	74.18	23.44
$\ell_1$ -OLS	69.32	68.21	20.45
$\ell_1$ -NLL	68.69	67.27	19.02
$\ell_1$ -Brier’s	70.37	72.55	20.36
$\ell_2$ -OLS	73.20	72.16	22.71
$\ell_2$ -NLL	72.37	69.35	21.08
$\ell_2$ -Brier’s	75.72	74.11	24.84
KL-OLS	68.08	65.28	18.24
KL-NLL	65.51	65.90	19.66
KL-Brier’s	64.30	67.22	20.75

Table 5.2: SS-MTL Embedding Analogy Transferability

Since word similarity scores are not directly optimized when using maximum likelihood, it is not obvious this is a suitable objective for improving on the evaluation metric  $\rho_s$ . Therefore, we also consider Brier’s score, which can be considered as MSE for class probabilities. Indeed, in Table 5.3 we find that using Brier’s score for the annotations improves  $\rho_s$  for 4 of 6 datasets.

Meta-embeddings that are learned only using unsupervised methods (Equation 5.1) give  $\rho_s = 45.10$ , on Simlex, while the semi-supervised MTL approach produces the most noticeable performance gain with a dramatic increase of  $\rho_s = 74.73$ . Although datasets such as Simlex have made a clear distinction between word associations and true similarity, we find there is still performance improvements made on true similarity

<sup>1</sup> Brier’s score [45] is a score between probabilistic predictions and is equivalent to MSE for regression.

	Simlex	WS353	RG	MTurk	RW	MEN
Cosine-OLS	53.63	73.13	83.07	69.41	60.49	80.25
Cosine-NLL	59.22	76.09	80.45	70.43	61.31	82.49
Cosine-Brier's	63.72	80.21	89.54	83.45	70.76	84.14
$\ell_1$ -OLS	55.16	68.80	82.82	70.35	61.07	78.56
$\ell_1$ -NLL	53.54	77.82	82.09	73.12	64.46	79.12
$\ell_1$ -Brier's	68.78	77.60	87.44	80.67	78.05	79.73
$\ell_2$ -OLS	68.31	73.85	84.48	70.91	53.20	81.60
$\ell_2$ -NLL	53.80	71.15	85.10	71.51	50.61	79.38
$\ell_2$ -Brier's	74.73	69.68	85.29	76.30	80.07	70.64
KL-OLS	62.47	68.93	85.75	72.35	50.38	80.95
KL-NLL	48.91	67.93	86.67	72.33	48.91	78.98
KL-Brier's	71.39	66.91	87.58	73.43	67.11	81.78

Table 5.3: Multi-Task Word Embedding Learning ( $\rho_s$ ) Results on Word Similarity

when transferring knowledge in the form of meta-embeddings from different annotation distributions that only score word association and not the true similarity [159].

In the semi-supervised MTL setting shown in Table 5.3, we see that results are also consistent with Table 5.1 as the cosine loss in reconstruction results in best performance for 4 out of the 6 datasets.

**ANALOGY RESULTS** We evaluate how the learned models from Table 5.3 transfer to analogy tasks, namely MSR Word Representation dataset [119] (8000 questions with 8 relations), Google Analogy dataset [283] (19,544 questions with 15 relations) and SemEval 2012 Task 2 Competition Dataset [188] (3218 question with 79 relations). The former two consist of categories of different analogy questions and the latter includes ranked candidate word pairs based on word pair relational similarity for a set of chosen word pairs. CosAdd [284] is used for calculating the analogy answers for Google and MSR which ranks candidates given as  $\text{CosAdd}(a, b, c, d) = \cos(b - a + c, d)$  and chooses the answer as the highest ranking candidate.

Table 5.2 shows the results of transferring the learned semi-supervised multi-task learning (SS-MTL) embeddings to analogy tasks.

Here, we analyse (1) whether the word meta-embeddings carry over to analogy even if not all embedding algorithms preserve analogy relations, (2) check if the similarity encoded with SS-MTL has any effect on performance on analogy and (3) check whether the nonlinearity induced by autoencoding, performs relatively well (somewhat counter-intuitively,

Task	Model	Skipgram		FastText		GloVe		LexVec		Meta	
NER	CNN	81.25	79.77	83.19	81.46	84.62	81.93	80.08	79.24	88.15	86.63
	LSTM	83.59	81.61	84.80	82.27	82.29	80.08	82.65	80.74	91.23	88.58
	GRU	82.14	80.59	83.94	82.08	84.72	81.80	83.19	81.47	90.79	88.38
	Highway	81.72	80.39	83.88	82.61	82.17	81.28	84.55	81.24	89.85	87.05
UDPoS	CNN	88.78	87.42	88.91	87.57	88.76	87.49	87.18	87.01	89.43	88.38
	LSTM	90.43	89.55	90.61	88.91	90.64	89.62	90.39	89.16	91.89	91.12
	GRU	89.86	88.23	89.24	88.44	89.17	88.23	89.02	88.17	90.72	90.59
	Highway	90.11	88.73	89.97	88.46	89.81	88.02	89.23	88.08	90.26	90.01
Sentiment	CNN	84.03		85.42		83.73		84.44		89.21	
	LSTM	86.50		87.21		86.48		85.17		92.38	
	GRU	85.43		87.69		84.73		85.01		91.75	
	Highway	82.39		85.31		84.21		84.58		89.73	

Table 5.4: Validation (left) & Test (right) Accuracy (%): NER, UDPoS & Sentiment Analysis (only test acc.)

given the existence of linear relationships between analogy pairs [6]) for analogy reasoning.

In general, SS-MTL that incorporates similarity scores has some transferability to analogy based on the scores provided by the aforementioned word similarity datasets. For Google Analogy, the larger of the three datasets with the smallest range of relation types, we find that the SS-MTL model that previously trained with Cosine-Brier’s loss functions shows the best performance overall. This is consistent with findings from Table 5.1 where the same model performs best for 4 of 6 word similarity datasets. This suggests that performing additional nonlinear meta-word encoding somewhat preserves the linear structures preserved in models such as skipgram and fasttext. Additionally, we find Brier’s score to perform particularly well based on  $\rho_s$  results.

### 5.5.2 Extrinsic Evaluation

For **NER** we use the New York Times NER recipe tagging task [133] that contains 17.5k recipes which accumulates to 67.5k steps, 142.5k tags. We use 16,622 sentences for **Universal Dependency PoS** [312] (254k words) that contain weblogs, newsgroups, emails and reviews, which are used to define universal dependencies for the English UD Treebank (15/02/2017 version 2.0). The trees are converted to Stanford Dependencies and manually corrected to universal dependencies, predominantly by single annotations.

IMDB Movie reviews dataset [268] is used for **Sentiment Analysis** where we predict positive and negative sentiments for 50k reviews, where both positive and negative classes are balanced and the train/test split is also 50-50. All reviews are filtered to have at least 30 reviews, reviews

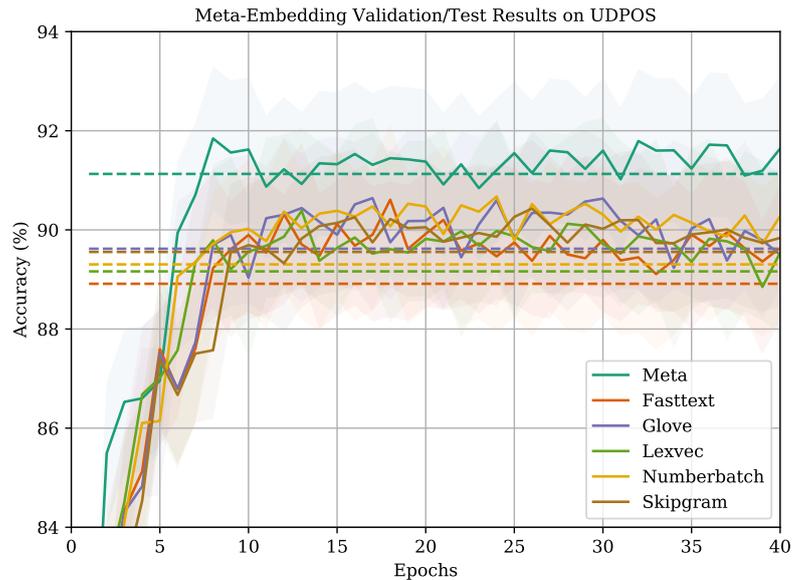


Figure 5.3: LSTM Results with Multi-Task Meta-Embedding on Universal Dependency PoS Tagging

were assigned as positive if the average score is 7/10 or higher, negative if  $\leq 4$  and neutral reviews are discarded.

### 5.5.3 Downstream Task Results

Table 5.4 shows the results on all 3 tasks, comparing the performance of each single pretrained embedding in the ensemble set to word meta-embeddings (Meta) that uses the reconstruction as an auxiliary task, as mentioned in Section 5.4. Based on the validation performance we set  $\lambda = 0.1$  for NER and UDPOS and  $\lambda = 0.15$  for Sentiment Analysis. Unlike the intrinsic tasks, we found a 2-hidden layer AE improved sentence-level meta embedding reconstruction for Sentiment Analysis. This is the only sentence classification dataset and therefore we use sentence-level meta-embedding regularization as opposed to word-level meta-embeddings that are used for intrinsic tasks and the remaining extrinsic tasks (NER and UDPOS tagging).

Improvements are found using meta-embedding reconstruction as an auxiliary task, when compared to using single pretrained embeddings. Overall, best results are obtained using meta-embeddings with an LSTM for the main task, while remaining models all show an increase in validation and test accuracy. Figure 5.3 shows results for UDPOS tagging where meta-embeddings have led to better results than using pretrained embeddings in multi-task learning. We find that near convergence the accuracy deviation in the validation set is lowered when using meta-embeddings, improving stability and calibrated probability estimates on the supervised tasks. This is also found for NER and sentiment

classification across all models tested. Moreover, meta-embedding reconstruction greatly improves the performance early on (<10 epochs), which means that less training time is needed when the shared layer is forced to preserve information from the embedding set.

## 5.6 SUMMARY

This chapter described the proposed multi-task learning approach to learning word meta-embeddings as an auxiliary reconstruction task to improve predictions on a main intrinsic (e.g word similarity, analogy) or extrinsic (e.g PoS, NER) task whereby the meta-embedding layer is a shared representation between tasks. We find consistent improvements against baselines and also identify objective functions for meta-embedding reconstruction that lead to optimal performance on the main task. In doing so, we identified a meta-embedding target autoencoder that learns to project between different permutations of different embedding spaces within the ensemble set and use the the mean of the resulting latent representations as the meta-embedding.

We find performance increased significantly when using manually annotated scores from word similarity datasets in comparison to single word embeddings and unsupervised word meta-embedding approaches. We also find that angular-based loss functions are well suited for word meta-learning for both self-supervised learning and the proposed multi-task semi-supervised learning method, showing best results on 4 out of the 6 word similarity datasets in both cases. Most significant improvements were found on relatively difficult word similarity and association datasets such as Simlex and rare word, while still improving by a large margin on the remaining datasets. Moreover, we find slight improvements made when transferring the semi-supervised models for analogy tasks. Lastly, we find consistent improvement when using meta-embeddings as an auxiliary task for downstream tasks such as Named Entity Recognition, Sentiment Classification and Universal Dependency PoS tagging.

However, this is expected given that similarity scores are more general than specific word pair relation types and not all word embedding algorithms preserve analogical relations to the same degree.



---

## KNOWLEDGE DISTILLED REINFORCEMENT LEARNING

---

Thus far, this thesis has discussed compression techniques in the context of supervised and unsupervised learning. In this chapter we describe a knowledge distillation approach in the context of reinforcement learning for conditional text generation.

Task-specific scores are often used to optimize and evaluate the performance of conditional text generation systems. However, such scores are non-differentiable and cannot be used in the standard supervised learning paradigm. Hence, policy gradient methods are used because the gradient can be computed without requiring a differentiable objective. In this chapter, we argue that the current  $n$ -gram overlap-based measures that are used as rewards can be improved by using model-based rewards transferred from tasks that directly compare the similarity of sentence pairs. These reward models either output a score of sentence-level syntactic or semantic similarities between entire predicted and target sentences as the expected return, or for intermediate phrases as segmented accumulative rewards. We demonstrate that *Transferable Reward Learner* leads to improved results on semantic evaluation measures in policy-gradient models for image captioning tasks.

We find that our actor-critic model that is optimized for model-based *InferSent* rewards improves on an Word Mover’s Distance similarity evaluation measure by 6.79 points on MSCOCO when compared to using the same model optimized for BLEU scores. Similarly, we find a further 10.48 point increase over BLEU optimized actor-critic models when evaluated on Sliding Window Cosine Similarity measure.

Improvements are also obtained on the smaller Flickr-30k dataset, demonstrating the general applicability of the proposed transfer learning method.

### 6.1 INTRODUCTION

Neural network based encoder-decoder architectures are increasingly being used for conditional text generation given the recent advances in CNNs [406] and RNNs [165], the latter of which use internal gating mechanisms to preserve long-term dependencies [165] and have shown

impressive results for density estimation (i.e language modelling) and text generation.

These encoder-decoder networks are usually trained end-to-end using ML training with full supervision (i.e the model learns explicitly from expert demonstrations). This is also referred to as *teacher forcing* [405].

Typically for text generation tasks, such as image captioning, CNNs [221] encode an image, which is passed to the beginning of the decoder RNN language model via a linear map from the image encoding. The decoder policy  $\hat{\pi}$  then performs a set of actions given an *expert* policy  $\pi^*$  for  $T$  time steps (e.g human captions). However, using full supervision using ML can act as a poor surrogate loss [405] for a task-specific score we are interested in and evaluate on (e.g BLEU). Moreover, these scores are non-differentiable and hence cannot be used in the standard supervised learning paradigm.

Deep Reinforcement Learning (DRL) can be used to optimize for task scores as rewards [19, 485], for generating higher quality texts in NLP tasks such as ROUGE-L [239] and BLEU [324], respectively used for evaluating summarization and machine translation systems.

Actor-critic networks in particular have reported State of The Art (SoTA) results for image captioning [19, 485]. These models use a policy network that produces actions, which are evaluated by a value network that outputs scores given both actions and targets as the input at each state. The values predicted by the critic are then used to train the actor network, assuming the critic values are exact (pre-training the critic is often necessary).

However, parameter-free measures such as BLEU and ROUGE-L do not correlate well with human judgements when compared to using learnable embedding-based evaluation measures on both word and sentence-level [197, 248, 411]. Moreover the frequency distribution of generated captions is significantly different to human captions [76].

We argue that  $n$ -gram overlap based measures in DRL models can be improved using model-based reward estimators that are transferred from sentence similarity models that directly learn from the human-annotated similarity ratings for sentence pairs. In the context of DRL, we view this type of transfer learning as generating the environment of a target task given a source model that implicitly learns relationships on a pairwise source task (i.e sentence similarity). Additionally, the reward is continuous everywhere and out-of-vocabulary terms do not hinder the TRL model’s ability to estimate rewards because sentence similarity can still be inferred even when  $\langle \text{unk} \rangle$  tags are used at test time to replace tokens we have not seen at training time.

We are further motivated by the fact that transfer learning for text generation has already been successfully demonstrated using pretrained ImageNet CNN encoders [210]. However, to the best of our knowledge, transfer learning with respect to the decoder of DRL-based encoder-decoder models has been unexplored until this point.

We also note that for the common use of **DRL** in games and robotics, transfer learning is often made difficult since the environments and dynamics are often distinctly different from one another (e.g. games usually do not have the same states, actions, transition probabilities and rewards i.e. Markov Decision Process (MDP)). In contrast, the MDP for natural language is defined by the vocabulary used for a given corpus. Thus, given a sufficient amount of text the MDP for all corpora converge.

**CONTRIBUTIONS** This work proposes to transfer pairwise models that have been trained to learn a similarity score between various universal textual representations. These models are trained on a set of sentence-pair learning problems such as semantic textual similarity (STS) and natural language inference (NLI).

Herein, we refer to this as *Transferable Reward Learning* (TRL), a method that incorporates model-based reward shaping to improve task-specific scores in relation to semantic similarity as a measure of language generation quality.

We compare both unsupervised and supervised TRL models against **ML** training and previously proposed actor-critic models with model-free rewards such as BLEU and ROUGE-L. To the best of our knowledge, this is the first work that focuses on learning to transfer model-based rewards in sequence prediction.

## 6.2 METHODOLOGY

In this section, we describe the image captioning setup, standard policy-gradient training of sequence predictors and our proposed transfer reward learner that builds upon the actor-critic network using knowledge distillation.

### 6.2.1 Image Caption Setup

For an image  $I$  there is a corresponding caption sequence  $Y$  that contains tokens  $Y = (y_1, \dots, y_T)$  and  $y \in \mathcal{V}$  where  $\mathcal{V}$  is the vocabulary. The encoder,  $f_\omega$ , encodes an image  $I$  into a hidden state  $z$  that is then passed to an RNN decoder  $f_\psi$  that generates a predicted sequence,  $\hat{Y}$ , which is then evaluated using a task-specific score  $R(Y, \hat{Y})$ . In the **DRL** setting, we can consider the problem as a finite MDP where each word  $w \in \mathcal{V}$  is considered a state  $s \in \mathcal{S}$  and a prediction  $\hat{y}_t$  is considered as an action  $\pi_\theta(a_t|s_t)$  in action space  $a \in \mathcal{A}$  with probability  $p_{\pi_\theta}(a_t|s_t)$ .

The environment then issues a discounted return,  $g = \sum_{t \in T} \gamma^t r_t$ , where the discounting factor,  $\gamma \in [0, 1]$ , after receiving the set of actions and the objective is then to maximize the total expected return  $G$ . We use an actor-critic model [25] as the basis of our experiments with a ResNet-152 encoder [151] and an **LSTM** decoder network.

### 6.2.2 Policy Gradient Training

We define a policy network,  $\pi_\theta$ , as an encoder-decoder architecture that encodes an image  $I_s \in \mathbb{R}^{m \times n}$  as  $h_s \in \mathbb{R}^n$  through the Resnet-152 [151] CNN-based encoder and a linear projection  $W_s \in \mathbb{R}^{d \times n}$  as shown in Equation 6.1. This is then concatenated with the embedding,  $x_t \in \mathbb{R}^m$ , corresponding to the input word,  $w_t \in \mathbb{Z}$ , which forms state  $s_t = x_t \oplus h_s$  where  $\oplus$  denotes concatenation. This LSTM decoder takes  $(s_t, h_{t-1})$  as input, as shown in Equation 6.2, omitting  $t = 0$  where  $h_0$  is used instead. Therefore, the policy network parameters include the ResNet-152 parameters ( $\omega$ ), the linear projection ( $W_s$ ), the LSTM parameters ( $\psi$ ) and the decoder projection layer ( $W_t$ ), i.e.  $\theta := \{\omega, W_s, \psi, W_t\}$ . The predictions for a given sequence length of  $T = |Y|$ , are defined as  $\hat{Y} = \{a_1, ..a_t, \dots, a_T\}$ , where the action space  $a_t \in \mathcal{A}$  is defined by the vocabulary  $w \in \mathcal{V}$  and the targets  $Y = \{w_1, .., w_t, \dots, w_T\}$ .

$$h_s = W_s \text{ResNet-152}(I_s) \quad (6.1)$$

$$h_t = \text{LSTM}(s_t, h_{t-1}) \quad (6.2)$$

$$p_{\pi_\theta}(s_t) = \phi(W_t h_t) \quad (6.3)$$

$$\pi_\theta(a_t | s_t) = p_{\pi_\theta}(a_t | s_t) \quad (6.4)$$

VALUE FUNCTION APPROXIMATION: For Value Function Approximation (VFA), the gradient of the expected cumulative discounted reward is typically estimated as  $\mathbb{E}[\sum_{t=0}^T \gamma^t r_t | a_{t+1}, ..a_T]$ , for reward  $r_t$  at time  $t$  for an  $l$ -step return.

We then use a critic network to estimate the state-value function that takes the policy,  $\pi$ , actions,  $a_{t:t+l} \in \hat{Y}$ , parameterized rewards,  $r_{t+1:t+l}^\vartheta$ , and compute the expected return,  $V^\pi(s_t)$ , from state,  $s_t$ , for  $l$  steps. This is given as the expectation over the sum of discounted rewards by Equation 6.5 where rewards  $r^\vartheta$  are issued by our proposed TRL model-based reward with frozen parameters  $\vartheta$  and  $\gamma$  not used as discounted rewards are not applicable in the TRL.

$$V^\pi(s_t) = \mathbb{E} \left[ \sum_{t=0}^l r_{t+1}^\vartheta | a_{t+1}, ..a_l \right] \quad (6.5)$$

ADVANTAGE FUNCTION APPROXIMATION: Above, we considered using  $V^\pi(s_t)$  to estimate  $g$ . However, training the value network from scratch can result in high-variance in the gradient resulting in poor convergence. Following Zhang et al. [485], we use the Advantage Function Approximator (APA),  $A^\pi$ , to reduce the variance in gradient updates. This is achieved using temporal-difference learning (TD- $\lambda$ ) as shown in Equation 6.6 where the Q-function is  $Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}:T, a_{t+1}:T} \left[ \sum_{i=0}^T r_{t+i} \right]$ , for an  $N$  step expected return

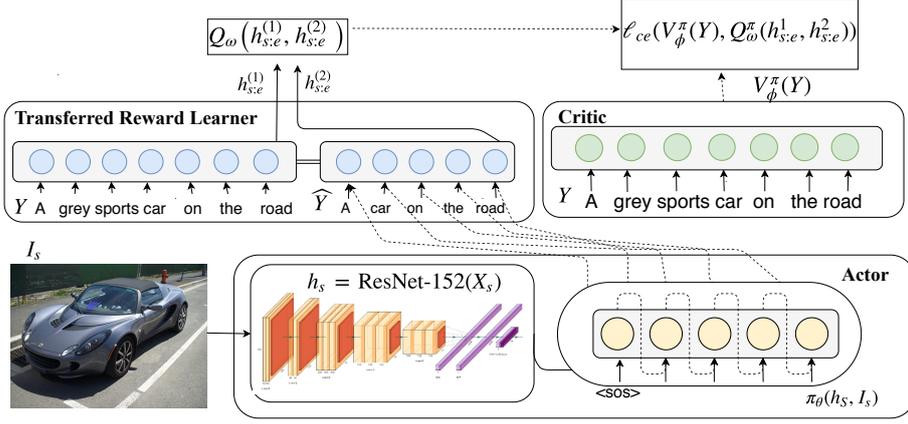


Figure 6.1: Actor  $\pi_\theta(\cdot)$  produces actions  $\hat{Y}$  given an encoded image  $h_s$  and caption  $X_t$  that are passed to  $\text{TRL}(Y, \hat{Y})$  and encoded into  $(h_1, h_2)$  respectively and scored with action-value  $A^\pi = V_\Theta^\pi(s) - Q^\pi(s, a)$

$G_t^i$  and  $\lambda$  is the trace decay parameter  $0 \leq \lambda \leq 1$  (larger  $\lambda$  values assign more credit to distant rewards).

$$\begin{aligned} A^\pi(s_t, a_{t+1}) &= Q^\pi(s_t, a_{t+1}) - V^\pi(s_t) \\ &= (1 - \lambda) \sum_{i=1}^N G_t^i - V^\pi(s_t) \end{aligned} \quad (6.6)$$

The gradient of the policy network can then be rewritten as in [Equation 6.7](#). Here, the trace decay parameter is set to  $0 < \lambda < 1$ , in our experiments  $\lambda = 1$  which corresponds to Monte-Carlo and means that large traces are also assigned to distant states and actions.

In the context of image captioning, it is typical that the episodes are short ( $T < 30$ ) and hence it is feasible.

$$G = \mathbb{E} \left[ \sum_{t=0}^{T-1} \left( (1 - \lambda) \sum_{i=0}^N G_t^m - V^\pi(s_t) \nabla_\theta \log \pi_\theta(a_{t+1}|s_t) \right) \right] \quad (6.7)$$

This is achieved by computing the gradient of the log likelihood multiplied by the advantage function,  $A^\pi(s_t, a_{t+1})$ , shown in [Equation 6.8](#). Here,  $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$  reduces the variance of the of the gradient by increasing the probability of actions when  $A^\pi(s_t, a_{t+1}) > 0$  and decrease otherwise.

$$G := \mathbb{E} \left[ \sum_{t=0}^{T-1} A^\pi(s_t, a_{t+1}) \nabla_\theta \log \pi_\theta(a_{t+1}|s_t) \right] \quad (6.8)$$

### 6.2.3 Transfer Reward Learner

We now consider two sentence encoders for TRL. We note that, although there has been considerable breakthroughs in recent years for models

that could be used for sentence similarity tasks [90], these models are too computationally costly to consider for issuing rewards and typically have more parameters than the whole actor-critic network combined. Hence, we focus on relatively modest sized pairwise models for training on the sentence similarity task. Both TRL models that evaluate state-action pairs are denoted as  $R_{\vartheta}(s, a)$  where the  $\vartheta$  parameters are not-updated as rewards are kept fixed throughout training. The advantage of this is that we are not restricted to selecting  $\lambda = 1$ , which is used for the sentence-level  $n$ -gram overlap measures such as BLEU, ROUGE and CIDEr. The critic can evaluate partially generated sequences and sentence pairs of different lengths, because the critic has been trained to learn similarity between sentences of non-equal lengths. We emphasize at this point that the TRL is not updated for value function estimation in our experiments, this is only carried out for approximating the advantage and value functions.

**INFERSENT REWARDS** For reward shaping, we use a pretrained sentence similarity model such as InferSent, tuned on SemEval 2017 Semantic Textual Similarity (STS) dataset<sup>1</sup> consisting of English monolingual sentence-pairs that are labelled with a score from 0 (semantic independence) to 5 (semantic equivalence).

The scores are scaled from the continuous [0-5] range to [0-1] using a sigmoid  $\sigma(\cdot)$  to convert to a probability.

Conneau et al. [70] used the scoring function given by Equation 6.9 between two encoded sentence-pairs,  $(h_1, h_2)$ , where  $h_1, h_2 \in \mathbb{R}^d$ , corresponding to the two sentences respectively  $\mathcal{S}_1$  and  $\mathcal{S}_2$ . We also use this scoring function for the pretrained InferSent model.

$$\phi\left([h_1, h_2, |h_1 - h_2|, h_1 \cdot h_2] \cdot W + b\right) \quad (6.9)$$

InferSent uses a Bidirectional-GRU (or BiLSTM) with max (or mean) pooling, as in Equation 6.10 where  $g$  represents the pooling function and  $e_i$  is the embedding corresponding to word  $x_i$ .

$$h = g_{\max\text{-pool}}\left(\left[\overrightarrow{\text{GRU}}(e_1, \dots, e_T), \overleftarrow{\text{GRU}}(e_1, \dots, e_T)\right]\right) \quad (6.10)$$

We also use the self-attentive variation in Equation 6.11, where the max-pooling operation  $g$  is replaced with self-attention that produces a weighted average where  $g_{\text{avg}}(\cdot)$  sum the weights to 1  $\forall t \in T$ .

Hence, attention focuses on the hidden states of important tokens prior to using the scoring function.

$$h = \sum_{t=1}^T g_{\text{avg}}(\tanh(Wh_t + b))h_t \quad (6.11)$$

<sup>1</sup> <http://alt.qcri.org/semeval2017/task1/>

**SKIPTHOUGHT REWARDS** We also consider the Skipthought as an unsupervised sentence representation method, which has shown competitive performance on sentence-level pairwise tasks [204]. This allows us to compare against the supervised sentence embeddings produced by InferSent. Similarly to the InferSent model, we use the same scoring function given by Equation 6.9.

**CRITIC LOSS** Prior work has used an  $\ell_2$  loss between policy values ( $\pi_\theta(a|s)$ ) and the critic scores ( $Q^\pi(s, a)$ ) [485]. In our preliminary experiments, we found the KL-divergence loss to outperform an  $\ell_2$  loss.

Therefore, we minimize the KL divergence given by Equation 6.12 between the  $[0, 1]$  normalized  $\tilde{Q}^\pi(s_t, a_{t+1})$  and  $\tilde{V}_\Theta^\pi(s_t)$ , which corresponds to minimizing the cross-entropy loss when ignoring  $\mathcal{H}(\tilde{Q}^\pi(s_t, a_{t+1}))$  that does not depend on  $\Theta$ .

$$\ell_{ce} = \mathcal{D}_{\text{KL}}\left(\tilde{Q}^\pi(s_t, a_{t+1}) \parallel \tilde{V}_\Theta^\pi(s_t)\right) + \mathcal{H}\left(\tilde{Q}^\pi(s_t, a_{t+1})\right) \quad (6.12)$$

The logit penalizes values that are much higher than the baseline more than the  $\ell_2$  loss but shows a larger gap between small value improvements over the baseline. This has the effect of stabilizing the critic network over consecutive iterations, as the critics gradient updates are not as large, stabilizing the training of the critic and subsequently ensuring the difference in the policy network  $\pi_\theta(a_t|s_t)$  is less drastic between iterations.

## 6.3 EXPERIMENTAL SETUP

This section describes the datasets used, training details, previous state of the art results and how embedding similarity between generated sentences and the corresponding ground truth sentences are evaluated. This will provide the necessary information before presenting the results on transfer reward learning.

### 6.3.1 Dataset Details

We use the Microsoft *Common Objects in Context* (MSCOCO) 2014 image captioning dataset proposed by Lin et al. [244], which is the de-facto benchmark for image captioning. This dataset includes 164,062 images (82,783 training images, 40,504 validation images, and 40,775 test images) with 5 manually labelled captions per image of 80 object categories and 91 *stuff* categories. Each image is paired with as least five manually annotated captions. We also use the smaller Flickr-30k [472] dataset, which contains 30k images with 150k corresponding captions, which also includes a constructed denotation graph that can be used to define denotational similarity, giving more generic descriptions through lexical and syntactic operations.

### 6.3.2 Training Details

As mentioned before, we use the ResNet-152 [151] classifier trained on ImageNet as our encoder. The reported experimental results are that of a 2-hidden layer LSTM decoder [165] network, with embedding input size and hidden layer size of  $|e| = |h| = 512$ . For both MSCOCO and Flickr30k we use mini-batches of size  $|x_{sub}| = 80$  with adaptive momentum (adam) [202] for training the LSTM decoder while, the image encoder is kept fixed in our experiments.

Training both actor and critic networks from scratch is difficult for policy gradient algorithms because it is often the case that the reward signal leads to high variance during the gradient updates, particularly in the early phase of training where the parameters  $\theta$  and  $\phi$  are initialized randomly. Therefore, in all our experiments we pre-train the actor and critic networks following Ren et al. [357] respectively for 5 and 7 epochs by minimizing the cross entropy loss  $-\sum_{t=1}^T \log p_{\theta}(a_t|s_t)$ . After the actor is pre-trained, the critic network is passed sampled actions from the fixed pre-trained actor and updated accordingly. After this initial phase, we then begin training both actor and critic together.

Methods	B1	B2	B3	B4	PPL
<b>NeuralTalk</b> Karpathy and Fei-Fei	0.57	0.37	0.24	0.16	-
<b>Mind’s Eye</b> Chen and Lawrence Zitnick	-	-	-	0.13	19.10
<b>NIC</b> Vinyals et al.	0.66	-	-	-	-
<b>LRCN</b> Donahue et al.	0.59	0.39	0.25	0.17	-
<b>m-Rnn-AlexNet</b> Mao et al.	0.54	0.36	0.23	0.15	35.11
<b>m-Rnn-VggNet</b> Mao et al.	0.60	0.41	0.28	0.19	20.72
<b>Hard-Attention</b> Xu et al.	0.67	0.44	0.30	0.20	-
Liu et al.					
<b>Implicit-Attention</b>	-	-	0.29	0.19	-
<b>Explicit-Attention</b>	-	-	0.29	0.19	-
<b>Strong Sup</b>	-	-	30.2	21.0	0.19
Wu et al.					
<b>Att-GT+LSTM</b>	0.78	0.57	0.42	0.30	14.88
<b>Att-SVM+LSTM</b>	0.68	0.49	0.33	0.23	16.01
<b>Att-GlobalCNN+LSTM</b>	0.70	0.50	0.35	0.27	16.00
<b>Att-RegionCNN+LSTM</b>	0.73	0.55	0.40	0.28	15.96

Table 6.1: SoTA Methods for Flickr30k

### 6.3.3 Embedding Similarity Evaluation

**WORD MOVER’S DISTANCE SENTENCE SIMILARITY** We also include WMD [212] for measuring semantic similarity between  $\ell_2$  normalized embeddings associated with predicted and target words. Word-level embedding similarities offer a faster alternative to model-based sentence-level evaluation, hence we include it for our experiments. To align WMD with word overlap metrics, we also include the penalization terms such

Flickr-30k		ROUGE-L		BLEU2		BLEU3		BLEU4		WMD		COS	
		Val.	Test	Val.	Test	Val.	Test	Val.	Test	Val.	Test	Val.	Test
Baseline	<b>ML</b>	33.09	31.46	67.52	65.20	42.12	40.68	27.81	26.87	65.35	63.62	60.14	59.22
	<b>BLEU</b>	35.68	32.93	70.03	70.39	48.65	48.45	30.88	30.79	73.99	72.70	66.22	66.10
	<b>ROUGE-L</b>	36.47	33.67	68.98	68.55	47.55	47.14	31.68	31.56	71.94	70.08	65.55	62.42
Our	<b>WMD</b>	31.61	30.76	70.06	68.05	46.24	44.23	29.78	28.94	76.59	73.40	69.83	68.73
	<b>InferSent</b>	30.08	29.29	69.48	68.76	44.76	43.96	28.79	28.70	<b>78.28</b>	<b>77.01</b>	<b>71.23</b>	<b>69.16</b>
	<b>Skipthought</b>	26.22	25.36	70.01	67.43	43.41	42.20	29.23	27.81	<b>76.50</b>	<b>75.18</b>	<b>72.63</b>	<b>68.60</b>

Table 6.2: Flickr30k Results for ML, Actor-Critic and our proposed TRL AC Models (using a beam width of 5)

as the brevity penalties used in BLEU [324], as shown in Equation 6.13. Here,  $\gamma$  is the similarity measure, the length ratio  $lr = |Y|/|\hat{Y}|$  [387] and the brevity penalty  $bp = \min(\exp(1 - 1/lr), 1)$ , which penalizes shorter length generated sentences.

$$s = \sigma \left( bp \cdot \gamma_{wmd}(E_{\hat{Y}}, E_Y) \right) \quad (6.13)$$

**SLIDING KERNEL COSINE SIMILARITY** We also considered decayed  $k$ -pairwise cosine similarity where  $k$  is a sliding window span that compares embeddings corresponding to  $n$ -gram groupings with a decay factor  $\gamma \in [0, 1]$  that depends on the distance such that  $\gamma_{(i,j)} = d(Y_i, Y_j)/k \forall i, j \in T$ .

Specifically, we use the kernel  $\gamma = \exp(-||i - j||)$  where  $i$  is the index corresponding to  $y \in Y$  and  $j$  for  $\hat{Y}$  respectively. This allows for any mis-alignments between sentences, as some may be shorter than others. There are  $T/k$  window spans, therefore we multiply the  $k/T$  by the brevity penalty.

$$s_{kcos} = \sigma \left( \frac{k}{T} BP \sum_{i=1}^T \sum_{j=i-k}^{i+k} \gamma_{(i,j)} \cos(E_{Y_i}, E_{\hat{Y}_j}) \right)$$

$$s.t., \quad t \leq i \leq T - k \quad (6.14)$$

## 6.4 RESULTS

### 6.4.1 Flickr30k

Table 6.1 shows the SoTA for image captioning on the Flickr30k dataset, not specific to policy-gradient methods as not all relevant papers include Flickr30k in experiments. Models proposed by Wu et al. [449] incorporate external knowledge (SPARQL queries over DBpedia knowledge base) for image captioning, hence the increase in BLEU and Perplexity (PPL). Table 6.2 compares ML training with previously published actor-critic approaches that use BLEU and ROUGE as the reward signal [485].

MSCOCO		ROUGE-L		BLEU1		BLEU2		BLEU3		BLEU4		CIDEr		METEOR		WMD		COS		
		Val.	Test	Val.	Test	Val.	Test	Val.	Test	Val.	Test									
SoTA	MIXER [352]	-	53.8	-	-	-	-	-	-	-	-	30.9	-	101.9	-	25.5	-	-	-	-
	MIXER-BCMR [352]	-	53.2	-	72.9	-	55.9	-	41.5	-	30.6	-	92.4	-	24.5	-	-	-	-	-
	PG-BCMR [352]	-	55.0	-	75.4	-	59.1	-	44.5	-	33.2	-	101.3	-	25.7	-	-	-	-	-
	SPIDER [252]	-	54.4	-	74.3	-	57.9	-	43.1	-	31.7	-	100.0	-	25.1	-	-	-	-	-
	RAML @ $r=0.9$ [267]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	VSE@A - 0.4 [357]	-	52.5	-	71.3	-	53.9	-	40.3	-	30.4	-	93.7	-	24.7	-	-	-	-	-
	TD-AC [485]	-	<b>55.4</b>	-	<b>77.8</b>	-	<b>61.2</b>	-	<b>45.9</b>	-	<b>33.7</b>	-	<b>116.2</b>	-	<b>26.7</b>	-	-	-	-	-
	SCST [338]	-	54.3	-	-	-	-	-	-	-	31.9	-	106.3	-	25.5	-	-	-	-	-
	SCST [449]	-	54.3	-	-	-	-	-	-	-	31.9	-	106.3	-	25.5	-	-	-	-	-
	ML	51.39	50.28	72.73	69.09	49.70	49.33	31.89	31.45	24.09	23.67	84.93	84.03	23.68	23.45	72.89	71.46	71.01	70.55	
Baselines BLEU	52.75	52.01	75.91	74.17	61.34	61.72	47.91	46.58	35.09	34.57	94.46	93.41	25.55	25.27	74.38	73.09	73.39	71.86		
ROUGE-L	56.28	55.25	72.98	72.55	51.55	50.29	37.44	35.38	32.44	31.09	95.53	95.51	25.61	25.54	73.53	72.65	73.88	72.10		
WMD	51.61	52.05	73.01	72.81	52.33	52.70	39.17	38.24	32.74	30.09	99.03	98.46	27.12	27.09	79.12	78.42	80.07	78.72		
Proposed InferSent	<b>55.46</b>	<b>54.25</b>	<b>75.58</b>	<b>75.02</b>	<b>60.40</b>	<b>57.16</b>	<b>46.24</b>	<b>41.68</b>	<b>31.93</b>	<b>31.24</b>	<b>106.12</b>	<b>105.68</b>	27.31	27.18	<b>82.86</b>	<b>80.26</b>	<b>83.71</b>	<b>82.58</b>		
Proposed Skiphought	53.02	52.71	74.49	73.61	54.54	51.08	32.28	31.02	29.78	28.59	105.56	105.08	<b>27.42</b>	<b>27.20</b>	81.95	80.21	81.25	80.63		

Table 6.3: MSCOCO Results for ML, Actor-Critic and our proposed TRL AC Models (B=5)

We use beam search with a beam size of  $B = 5$  at test time. The beam search retains  $B$  most probable prediction at each timestep and considers the possible next token  $w_{t+1}^b$  extensions for a beam  $b$  and repeats until timestep  $T$ ,  $\forall b \in B$ .

When using only WMD as the reward signal, which is model-free, we see that there are improvements on semantic similarity measures (i.e WMD and COS). Here, COS refers to the Sliding Kernel Cosine Similarity described in the previous section.

Interestingly, we also find WMD improves over ML for word-overlap despite WMD not optimizing for a Dirac distribution, like ML training.

Both TRLs (InferSent and Skiphought) make significant improvements on WMD and COS. Hence, we infer that these TRLs that learn sentence similarity produce semantically similar sentences at the expense of a decrease in word-overlap (expected since the model is not restricted to predicting the exact ground truth tokens). This relaxes the strictness of word-overlap and allows for diversity in the generated captions. Moreover, WMD does not penalize sentence length and thus promotes diversity in caption length. However, as mentioned, we do include brevity penalty in WMD and COS for the purposes of easier comparison to word overlap metrics.

#### 6.4.2 MSCOCO

The top of Table 6.3 shows SoTA results for policy-gradient methods based on the best average score on BLEU [324], ROUGE-L [239], METEOR [85], CIDEr [425] evaluation metrics. VSE is the aforementioned Visual Semantic Embedding model that uses TD( $\lambda$ ) at  $\lambda = 0.4$ . For SCST, these results are from the test portion of the Karpathy splits [192] using CIDEr for optimization. Policy gradient methods have reached near top of the MSCOCO competition leaderboard without using ensemble models.

The lower end shows the results of our proposed models and baselines evaluated on both  $n$ -gram overlap based measures and word-level (Cosine) and sentence-level (WMD) embedding based evaluation measures.

We find that the largest gap in performance between our proposed TRLs and n-gram overlap metrics (BLEU and ROUGE-L) reward signals are found on the embedding-based evaluation measures. For all TRLs (WMD, InferSent, Skipthought) performance consistently improves over ML, BLEU and ROUGE-L when evaluated on WMD and Cosine. This suggests that even though we may not strictly predict the correct word as measured by word-overlap measures, the semantic similarity of sentences is preserved as measured by WMD and Sliding Kernel Cosine Similarity. Furthermore, this results in more diverse text generation as the policy network is not penalized for constructing candidate sentences that do not have high word overlap with the reference captions. TRLs outperform word overlap policy rewards such as BLEU



Human	<p><i>An image of a cars driving on the highway</i></p> <p><i>A section of traffic coming to a stop at an intersection.</i></p> <p><i>A bunch of cars sit at the intersection of a street.</i></p> <p><i>This is a picture of traffic on a very busy street.</i></p> <p><i>A busy intersection filled with cars in asia.</i></p>
ML AC-BLEU	<p><i>an image of a sitting car in traffic</i></p> <p><i>A group of cars at an intersection.</i></p>
AC-WMD AC-Skipthought AC-InferSent	<p><i>A group of cars at lights near a traffic intersection.</i></p> <p><i>A group of cars near a busy intersection road.</i></p> <p><i>A picture of cars stopping near the traffic intersection.</i></p>

Figure 6.2: Qualitative Results on MSCOCO

and ROUGE-L on our embedding similarity based metrics. Of the three, we find the InferSent TRL to outperform the other two, with the unsupervised Skipthought TRL being competitive for all metrics. We also

see results are competitive to the SoTA. We find similar findings for TRL models evaluated on CIDEr and METEOR.

Figure 6.2 shows an example of the ground truth captions (Human), ML trained generated caption, a baseline AC trained with BLEU scores and our three proposed alternatives that improve for semantic similarity. We demonstrate the difference between text generated for an image of a traffic jam near an intersection. The example also illustrates that the ground truth itself is imperfect, both syntactically (‘.of a cars.’) and semantically (‘.cars sit at the intersection.’). The TRL will assign lower return in these cases, whereas word-overlap measures do not explicitly penalize how bad the semantic or syntactic differences are between predicted and ground truth sentences.

## 6.5 SUMMARY

In summary, this chapter described a reinforcement learned text generator that uses pretrained models that are specifically trained on sentence similarity tasks that can be used to issue rewards and to define, optimize and evaluate language quality for neural-based text generation. We find performance on semantic similarity metrics improve over a policy gradient model, namely the actor-critic model, that uses unbiased word overlap metrics as rewards. The InferSent actor-critic model improves over a BLEU trained actor-critic model on MSCOCO when evaluated on a Word Mover’s Distance similarity measure by 6.97 points and 10.48 points on sentence-level cosine embedding metric. Large performance gains are also found for Flickr-30k dataset, demonstrating the general applicability of the proposed transfer learning method. We conclude that model-based task should be considered for reinforcement learning based approaches to conditional text generation.

---

## KNOWLEDGE DISTILLED METRIC LEARNING

---

Negative Sampling (NS) is a limiting factor w.r.t. the generalization of metric-learned neural networks. We show that uniform negative sampling provides little information about the class boundaries and thus propose three novel techniques for efficient negative sampling: drawing negative samples from (1) the top- $k$  most semantically similar classes, (2) the top- $k$  most semantically similar samples and (3) interpolating between contrastive latent representations to create pseudo negatives. Our experiments on CIFAR-10, CIFAR-100 and Tiny-ImageNet-200 show that our proposed

*Conditioned Negative Sampling* and Latent Mixup lead to consistent performance improvements. In the standard supervised learning setting, on average we increase test accuracy by 1.52% percentage points on CIFAR-10 across various network architectures. In the knowledge distillation setting, (1) the performance of student networks increase by 4.56% percentage points on Tiny-ImageNet-200 and 3.29% on CIFAR-100 over student networks trained with no teacher and (2) 1.23% and 1.72% respectively over a *hard-to-beat* baseline [162].

### 7.1 INTRODUCTION

Training deep neural networks using contrastive learning has shown SoTA performance in domains such as computer vision [61, 150, 156, 321], speech recognition [321] and NLP [106, 260, 301]. The generalization performance in contrastive learning heavily relies on the quality of negative samples used during training to define the classification boundary and requires a large number of negative samples. Hence, contrastive learning relies on techniques that enable training of large batch sizes, such as learning a lookup table [150, 452, 453] to store lower-dimensional latent features of negative samples. However, training large models using contrastive learning can be inefficient when using uniformly sampled negatives (USNs) as the total number of potential negative sample pairs is  $\mathcal{O}((N - N_+)N)$  where  $N$  is the number of training samples and  $N_+$  is the number of positive class samples. Hence, even a lookup table may poorly estimate negative sample latent features, even for a large number of USNs and training epochs.

A complementary approach to improve the learning efficiency of a contrastive learned neural network is to reduce the model size using model compression techniques such as knowledge distillation [KD; 49]. In neural networks, this is achieved by transferring the logits of a larger “teacher” network to learn a smaller “student” network [162]. There has been various Knowledge Distillation (KD) methods proposed [162, 330, 333, 360, 419, 467, 478]. They involve minimizing KL divergence (KLD) between the student network and teacher network logits [162], minimizing the squared error between the student and teacher network intermediate layers [360], metric learning approaches [4, 328, 414, 419], attention transfer of convolution maps [478] and activation boundary transfer [157].

In this chapter, we discuss our proposed three efficient NS alternatives to uniform NS and show their efficacy in standard supervised learning and the aforementioned KD setting. Our NS techniques are also complementary to the aforementioned lookup tables used for retrieving negative latent features. All three techniques have a common factor in that they produce negatives that are semantically similar to the positive targets on both instance- and class-levels. We collectively refer to these sampling methods as Semantically Conditioned Negative Sampling (SCNS) as we replace a uniform prior over negative samples with a conditional probability distribution defined by the embeddings produced by a pretrained network. SCNS provides more informative negatives in contrast to USNs where many samples are *easy to classify* and do not support the class boundaries. Additionally, it requires no additional parameters at training time and thus can be used with large training sets and models. The pretrained representations are used to estimate pairwise class-level and instance-level semantic similarities to define a top- $k$  NS probability distribution. This reduces the number of negative pairs from  $\mathcal{O}(N(N - N_+))$  to  $\mathcal{O}(Nk)$  where  $k$  is the number of nearest neighbor negative samples for each sample. Below is a summary of our contributions.

**1) Class and Instance Conditioned Negative Sampling:** We define the NS distribution of each class by drawing negative samples proportional to the top- $k$  cosine similarity between pretrained word embeddings of the class labels. We also propose top- $k$  instance-level similarity for defining the NS distribution by performing a forward pass with a pretrained network prior to training.

**2) Contrastive Representation Mixup:** In Section 7.3.2, we propose *Latent Mixup* (LM), a variant of Mixup [484] that operates on latent representations between teacher positive and negative representations to produce harder pseudo negative sample representations that lie closer to the class boundaries. This is also carried out for the student network representations and a distance (or divergence) is minimized between both mixed representations.

**3) Theoretical Analysis of Conditioned Sampling:** The mutual information lower bound is reformulated to account for semantic similarity of contrastive pairs and describe the sample efficiency of SCNS compared to uniform sampling.

## 7.2 RELATED RESEARCH

Before describing our proposed methods, we review related work on the two most related aspects: efficient NS and KD.

**Efficient Negative Sampling** Efficient NS has been explored in the literature, predominantly for triplet learning. *Semi-hard* NS has been used to sample negative pairs yet are still further in Euclidean distance than the anchor-positive pair [379]. Oh Song et al. [320] combine contrastive and triplet losses to mine negative structured embedding samples, drawing negative samples proportional to a Gaussian distribution of negative sample distances to the anchor sample Harwood et al. [146]. Suh et al. [400] select hard negatives from the class-to-sample distances and then search on the instance-level within the selected class to retrieve negative samples. Wu et al. [446] proposed a distance weighted sampling that selects more stable and informative samples when compared to uniform sampling and show that data selection is at least as important as the choice of loss function. Zhuang, Zhai, and Yamins [497] define two neighborhoods using  $k$ -means clustering, *close* neighbors and dissimilar samples are *background* neighbors. Wu et al. [448] use ball discrimination to discriminate between hard and easy negative unsupervised representations where positive pairs are different views of the same image. Tran et al. [416] use the prior probability of observing a class to draw negative samples and then further sample instances within the chosen class based on the inner product with the anchor of the triplet, showing improvements over *semi-hard* NS without the use of informative priors.

**Knowledge Distillation** The original KD objective minimizes the Kullback-Leibler divergence between student network and teacher network logits [162]. Romero et al. [360] instead restrict the student network hidden representation to behave similarly to the teacher network hidden representations by minimizing the squared error between corresponding layers of the two networks. The main restriction in this method is that both networks have to be the same depth and of similar architectures. Attention Transfer (AT) [478] performs KD by forcing the student network to mimic the attention maps over convolutional layers of the pretrained teacher network. Passalis and Tefas [330] use Gaussian and Cosine-based kernel density estimators (KDEs) to maximize the similarity between the student and teacher probability distributions. They find consistent improvements over Hinton, Vinyals, and Dean [162] and *Hint layers* used in Fitnet [360]. Similarity-Preserving (SP) [419] KD ensures that the activation patterns of the student network are similar

to that in the teacher network for semantically similar input pairs. Peng et al. [333] propose Correlation Congruence (CC) to maximize multiple cross-correlations between samples of the same class. Ahn et al. [4] provide an information-theoretic view of KD by maximizing the mutual information between student and teacher networks through variational information maximization [24]. Moreover, Park et al. [328] argue that the distance between relation structures created from multiple samples of the student and teacher networks should be minimized. They propose Relational KD (RKD), which involves the use of both distance-wise and angle-wise distillation losses that penalize structural discrepancies between multiple instance outputs from both networks. Contrastive Representation Distillation [414] uses a CL objective to maximize a lower bound on the mutual to capture higher order dependencies between positive and negative samples, adapting their loss from Hjelm et al. [164].

### 7.3 METHODOLOGY

We begin by defining a dataset as  $\mathcal{D} := \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ , which consists of  $N$  samples of an input vector  $\mathbf{x} \in \mathbb{R}^n$  and a corresponding target  $\mathbf{y} \in \{0, 1\}^C$  where sample  $s_i := (\mathbf{x}_i, \mathbf{y}_i)$  and  $C$  is the number of classes. In the CL setting  $\mathbf{x} = (\mathbf{x}_*, \mathbf{x}_+, \mathbf{x}_{-,1}, \dots, \mathbf{x}_{-,M})$  where  $X_+ := (\mathbf{x}_*, \mathbf{x}_+)$  and  $X_- := (\mathbf{x}_*, \mathbf{x}_{-,1} \dots \mathbf{x}_{-,M})$  for  $M$  negative pairs. We denote a neural network as  $f_\theta(\mathbf{x})$  which has parameters  $\theta := (\theta_1, \theta_2, \dots, \theta_\ell, \dots, \theta_L)^T$  where  $\theta_l := \{\mathbf{W}_l, \mathbf{b}_l\}$ ,  $\mathbf{W}_l \in \mathbb{R}^{d_l \times d_{l+1}}$ ,  $\mathbf{b}_l \in \mathbb{R}^{d_{l+1}}$  and  $d_l$  denotes the dimensionality of the  $l$ -th layer. The input to each subsequent layer is denoted as  $\mathbf{h}_l \in \mathbb{R}^{d_l}$  where  $\mathbf{x} := \mathbf{h}_0$  and the corresponding output activation is denoted as  $\mathbf{z}_l = g(\mathbf{h}_l)$ . For brevity, we refer to  $\mathbf{z} = g(\mathbf{h}_L)$  as the *unnormalized output* where  $g : \mathbb{R}^{d_L} \rightarrow \mathbb{R}^p$  and  $\mathbf{z} \in \mathbb{R}^p$ . However, when using a metric loss,  $g : \mathbb{R}^{d_L} \rightarrow \mathbb{R}^{d_L}$  and therefore  $\mathbf{z} \in \mathbb{R}^{d_L}$ . In the former case, the cross-entropy loss is used for supervised learning and defined as  $\ell_{\text{CE}}(\mathcal{D}) := \frac{1}{N} \sum_{i=1}^N \sum_{c=1}^p -\mathbf{y}_{i,c} \log \hat{\mathbf{y}}_{i,c}$  where  $\hat{\mathbf{y}}_i = \sigma(f_\theta(\mathbf{x}_i)/\tau)$ ,  $\hat{\mathbf{y}}_i \in \mathbb{R}^p$  and  $\tau \in (0, +\infty)$  is the temperature of the softmax  $\sigma$ .

We also consider the KD setting where a student network  $f_\theta^S$  learns from a pretrained teacher network  $f_\omega^T$  with pretrained and frozen parameters  $\omega$ . The last hidden layer representation of  $f_\theta^S$  is given as  $\mathbf{z}^S := f_\theta^S(\mathbf{x})$  and similarly  $\mathbf{z}^T := f_\omega^T(\mathbf{x})$ . The Kullback-Leibler Divergence (KLD),  $D_{\text{KLD}}$ , between  $\mathbf{z}^S$  and  $\mathbf{z}^T$  is defined in Equation 7.1,

$$\begin{aligned} D_{\text{KLD}}(\mathbf{y}^T \parallel \mathbf{y}^S) &= \mathbb{H}(\mathbf{y}^T) - \mathbf{y}^T \log(\mathbf{y}^S) \\ \mathbf{y}^S &= \sigma(\mathbf{z}^S/\tau), \quad \mathbf{y}^T = \sigma(\mathbf{z}^T/\tau) \end{aligned} \tag{7.1}$$

where  $\mathbb{H}(\mathbf{y}^T)$  is the entropy of the teacher distribution  $\mathbf{y}^T$ . Following Hinton, Vinyals, and Dean [162], the weighted sum of cross-entropy

loss and KLD loss shown in Equation 11.1 is used as our main KD baseline, where  $\alpha \in [0, 1]$ .

$$\ell_{\text{KLD}} = (1 - \alpha)\ell_{\text{CE}}(\mathbf{y}^S, \mathbf{y}) + \alpha\tau^2 D_{\text{KLD}}(\mathbf{y}^S, \mathbf{y}^T) \quad (7.2)$$

To carry out KD using the KLD loss, the outputs of the pretrained teacher  $f_w^T$  are stored after performing a single forward pass over mini-batches  $\mathcal{B} \subset \mathcal{D}$  in our training set. These outputs are then retrieved for each mini-batch update of the smaller student network  $f_\theta^S$ . Given this background, the next two subsections will describe our three main approaches to improving NS in contrastive learning.

### 7.3.1 Conditioned Negative Sampling

Here, we describe two of our three approaches for improving NS efficiency that both involve using  $f_w^T$  to define a NS distribution. The first involves a cross-modal teacher network (i.e pretrained word embeddings for image classification) to define a **class-level** NS distribution and in the second  $f_w^T$  is a pretrained image classifier that defines an **instance-level** NS distribution.

#### 7.3.1.1 Class-Level Negative Sampling

Our first method assumes that word embedding similarity between class labels highly correlates with image embedding similarity [114, 229]. Pretrained word embedding similarities are used to improve the sample efficiency of NS in contrastive learning and replace uniform NS that is typically used. The cosine similarity is measured between the pretrained word embeddings  $(z_{w_i}^T, z_{w_j}^T)$  where  $(w_i, w_j)$  are the class labels in the vocabulary  $\mathcal{V}$  and  $|\mathcal{V}| = C$ . This is carried out for all pairs to construct an all pair cosine similarity matrix  $\mathbf{Z}_\mathcal{V} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$  that is then row-normalized with the softmax function  $\sigma$  as  $\mathbf{P}_\mathcal{V} := \sigma(\mathbf{Z}_\mathcal{V}/\tau)$ . Here, setting  $\tau$  high leads to harder negative samples being chosen from the most similar classes.  $\mathbf{P}$  represents the conditional probability matrix used to define  $X_-$  by drawing samples as Equation 7.3 where  $\mathcal{D}_w$

$$x_- \sim \mathcal{D}_w \propto \mathbf{P}_w \quad (7.3)$$

represents all samples  $(x_w^1, \dots, x_w^M)$  for a given class associated with  $w$ . This is repeated  $M$  times when using CL.

**HARD  $k$ -NEAREST CLASS-LEVEL NEGATIVE SAMPLES** Instead of sampling over a possible  $M = |\mathcal{V}| - 1$  number of negative samples, we can define the top- $k$  most similar *hard* negative samples. The top- $k$  cosine similarities from other labels in  $\mathcal{V}$  are selected by applying Equation 7.4 where  $z_{w_i}^T \in \mathbb{R}^{d_w}$  and  $z_{w_i} := f_\theta^T(w_i)$  of the class label  $w_i$ .

$$\text{topk}_w(\mathbf{z}_{w_i}^T) = \arg \max_{k \neq i} [\cos(\mathbf{z}_{w_i}^T, \mathbf{z}_{w_k}^T)] \quad (7.4)$$

The  $k$ -nearest neighbor ( $k$ -NN) similarity scores are then stored in  $\mathbf{Z}_{\mathcal{V}}^k \in \mathbb{R}^{|\mathcal{V}| \times k}$  with a corresponding matrix  $I_{\mathcal{V}}^k \in \mathbb{R}^{|\mathcal{V}| \times k}$  that stores the  $k$ -NN class indices  $\mathbf{Z}_{\mathcal{V}}^k$  retrieved by applying Equation 7.4. We focus on only sampling from the top- $k$  classes and therefore define the normalized top- $k$  class distribution matrix as  $\mathbf{P}_{\mathcal{V}}^k := \sigma(\mathbf{Z}_{\mathcal{V}}^k / \tau)$ . A row vector of  $\mathbf{P}_{\mathcal{V}}^k$  is denoted as  $\mathbf{P}_w^k$  consisting of the  $k$  truncated conditional probabilities corresponding to the  $k$  nearest class labels of  $w \in \mathcal{V}$ . We then sample as in Equation 7.3 instead with top- $k$  negative samples  $\mathcal{D}_w^k \subset \mathcal{D}_w$  and  $|\mathcal{D}_w^k| = k|\mathcal{D}_w|$ .

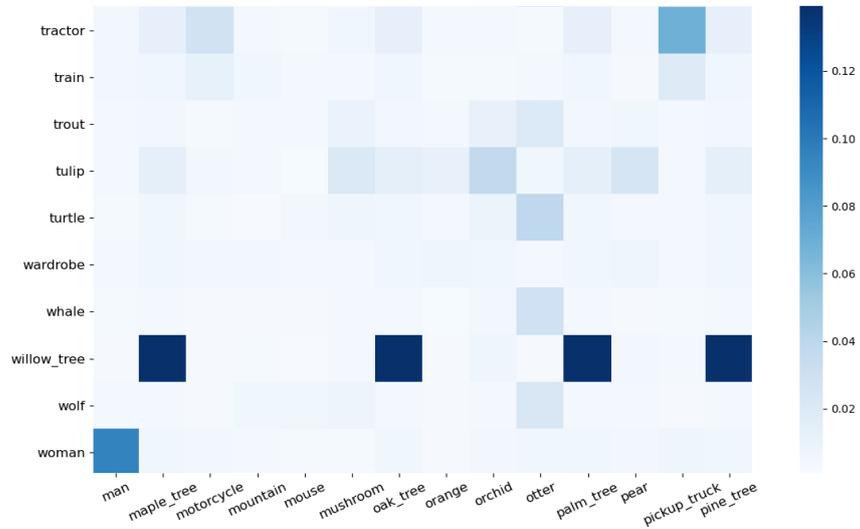


Figure 7.1: CIFAR-100 subset of word embedding class similarities

Figure (7.1) shows a submatrix of  $\mathbf{P}_{\mathcal{V}}$  as a heatmap corresponding to a subset of CIFAR-100 class labels on the x and y-axis. We see that “willow-tree” has a high similarity score with “maple-tree”, “oak-tree”, “palm-tree” and “pine-tree”. Therefore, samples from these class labels will be sampled more frequently as hard yet more informative negative samples. Similarly, (“man”, “woman”) and (“tractor”, “pickup-truck”) would be sampled at a higher rate than the remaining terms.

### 7.3.1.2 Instance-level Conditioned Sampling

Class-level SCNS (or Class-SCNS) may not be granular enough as the conditional probability assigned to a class is the same for all samples within that class. In instance-level SCNS we define the top- $k$  nearest samples for each  $\mathbf{x} \in \mathcal{D}$ . A top- $k$  instance similarity matrix produced by  $f_w^T$  is iteratively constructed  $\forall \mathbf{x} \in \mathcal{D}$  and the outputs are stored in  $\mathbf{Z}_{\mathbf{x}}^k \in \mathbb{R}^{N \times k}$ . As before, we define  $\mathbf{P}_{\mathbf{x}}^k$  and sample  $\mathbf{x}_- \sim \mathcal{D}_{\mathbf{x}}^k \propto \mathbf{P}_{\mathbf{x}}^k$ . Unlike Class-SCNS,  $f_w^T$  is trained on the same modality (e.g images)

as  $f_\theta^S$  and  $\mathbf{P}_x^k$  is now a SCNS matrix for each  $\mathbf{x}_* \in \mathcal{D}$  and not per class label  $w \in \mathcal{V}$ .

For image classification, we choose  $f_\omega^T$  to be a pretrained CNN. We use the final hidden representation  $\mathbf{z}_{x_*}^T \in \mathbb{R}^d$  which is a latent representation of an input image  $\mathbf{x} \in \mathbb{R}^{C_i \times d_{w_i} \times d_{h_i}}$  where  $C_i$  is the number of input channels,  $d_{w_i}$  is the width of the input image and  $d_{h_i}$  is the input image height. However, if the  $l$ -th intermediate layer activation  $\mathbf{h} \in \mathbb{R}^{C_o \times d_{h_o} \times d_{w_o}}$  is used to measure semantic similarity, we vectorize  $\mathbf{h}$  to  $\mathbf{h} \in \mathbb{R}^{C_o d_{h_o} d_{w_o}}$ , where  $C_o$  is the number of output channels,  $d_{h_o}$  is the height of the output feature maps and  $d_{w_o}$  is the output width. We note that Instance-SCNS may be favoured over Class-SCNS particularly when  $|\mathcal{V}|$  is relatively low and  $k \approx |\mathcal{V}|$  (e.g CIFAR-10).

### 7.3.1.3 Conditioned Samples with a Lookup Table

To further reduce training time we can combine SCNS with a lookup table [453] that stores negative sample features and updates during training. For  $\forall X_+ \in \mathcal{B}$ , the dot product is computed between  $\mathbf{z}^S$  and the  $i$ -th column of the  $L_2$  row normalized lookup table  $\mathbf{V} \in \mathbb{R}^{d_i \times N_v}$ . If the  $i$ -th target  $y_i$  is predicted then the update  $\mathbf{v}_i \leftarrow \gamma \mathbf{v}_i + (1 - \gamma) \mathbf{z}^S$  is performed in the backward pass. A high  $\gamma \in [0, 1]$  results in a smaller update  $\mathbf{v}_i \in \mathbb{R}^{d_i}$  in the lookup table. For  $X_-$ , a circular queue  $\mathbf{Q} \in \mathbb{R}^{d_i \times N_q}$  is used where  $N_q$  is the queue size and the dot product  $\mathbf{Q}^\top \mathbf{z}^S$  is computed. The current features are put into the queue and older features are removed from the queue during training. Equation 7.5 shows the conditional probability  $p_i$  corresponding to the target  $\mathbf{y}_i$  where  $\mathbf{u}_n \in \mathbb{R}^{d_i}$  is stored as the  $n$ -th row of  $\mathbf{Q}$ .

$$p_i = \frac{e^{(\mathbf{v}^\top \mathbf{z}_*^S / \tau)}}{\sum_{m=1}^{N_v} e^{(\mathbf{v}_m^\top \mathbf{z}_*^S / \tau)} + \sum_{n=1}^{N_q} e^{(\mathbf{u}_n^\top \mathbf{z}_*^S / \tau)}} \quad (7.5)$$

For the anchor sample  $\mathbf{x}_*$ , the conditional probability that the representation of a negative sample  $\mathbf{x}_-$  matches in the circular queue is given by Equation 7.6.

$$q_i = \frac{e^{(\mathbf{u}_i^\top \mathbf{z}_*^S / \tau)}}{\sum_{m=1}^{N_v} e^{(\mathbf{v}_m^\top \mathbf{z}_*^S / \tau)} + \sum_{n=1}^{N_q} e^{(\mathbf{u}_n^\top \mathbf{z}_*^S / \tau)}} \quad (7.6)$$

The gradient  $\nabla_{z^S} \mathbb{E}_{z^S} [\log p_i]$  is defined in the backward pass as Equation 7.7,

$$\frac{\partial \ell}{\partial \mathbf{z}^S} = \frac{1}{\tau} \left[ (1 - p_i) \mathbf{v} - \sum_{\substack{j=1, \\ j \neq y}}^{N_v} p_j \mathbf{v}_j - \sum_{k=1}^{N_q} q_k \mathbf{u}_k \right] \quad (7.7)$$

where  $y$  is the column of  $\mathbf{V}$  corresponding to the  $y$ -th target  $y \in \mathbb{N}_+$  and  $\mathbf{y}^T$  in the KD setting. This lookup table can be used complementary to SCNS and we use it in our experiments. It is also easier to compare to prior CL approaches [414] as they too use a lookup table.

---

**Algorithm 2** SCNS distillation algorithm.
 

---

```

1: input: mini-batch size  $M$ , number of batches  $N$ , student network
    $f_\theta$ , teacher network  $g_\varphi$ , regularization terms  $\gamma_+, \gamma_-$ .
2: for  $k$ -NN sampled minibatch  $\{\mathbf{B}_i\}_{i=1}^N$  do
3:    $\ell = 0$ 
4:   for all  $\{\mathbf{x}\}_{j=1}^M$  do
5:     # Positive embedding features
6:      $\mathbf{h}_*^S, \mathbf{h}_+^S, \mathbf{h}_-^S = f^S(\mathbf{x}_*), f^S(\mathbf{x}_+), f^S(\mathbf{x}_-)$ 
7:      $\mathbf{h}_*^T, \mathbf{h}_+^T, \mathbf{h}_-^T = f^T(\mathbf{x}_*), f^T(\mathbf{x}_+), f^T(\mathbf{x}_-)$ 
8:     # contrastive features retrieved from lookup table
9:      $\mathbf{z}_*^S, \mathbf{z}_+^S, \mathbf{z}_-^S = g^S(\mathbf{h}_*), g^T(\mathbf{h}_+), g^T(\mathbf{h}_-)$ 
10:     $\mathbf{z}_*^T, \mathbf{z}_+^T, \mathbf{z}_-^T = g^T(\mathbf{h}_*), g^T(\mathbf{h}_+), g^T(\mathbf{h}_-)$ 
11:    # Contrastive mixup representations
12:     $\tilde{\mathbf{z}}^S, \tilde{\mathbf{z}}^T = \kappa(\mathbf{z}_-^S, \mathbf{z}_*^S), \kappa(\mathbf{z}_-^T, \mathbf{z}_*^T)$ 
13:    # student network prediction
14:     $\mathbf{y}^S = \sigma(\mathbf{h}_*^S \mathbf{W}^T)$ 
15:    # Cross-entropy loss
16:     $\ell := \ell + (1 - \alpha) l_{\text{CE}}(\mathbf{y}^S, \mathbf{y})$ 
17:    # Latent Mixup Loss
18:     $\ell := \ell + \alpha l_{\text{KLD}}(\tilde{\mathbf{z}}^S, \tilde{\mathbf{z}}^T)$ 
19:    # KD loss of positive and negative samples
20:     $\ell := \ell - \gamma_+ l_{\text{KD}}(\mathbf{z}_+^S, \mathbf{z}_+^T)$ 
21:     $\ell := \ell - \gamma_- l_{\text{KD}}(\mathbf{z}_-^S, \mathbf{z}_-^T)$ 
22:   end for
23:   perform gradient updates on  $f_\theta$  to minimize  $\ell$ 
24: end for
25: return encoder network  $f(\cdot)$ , and throw away  $g(\cdot)$ 

```

---

### 7.3.2 Interpolating Contrastive Representations

Instead of using a pretrained network to define the negative samples that are close to the classification boundary, we can instead mix positive and negative representation to produce pseudo negative samples that are close to the positive sample. Mixup [484] is a simple regularization technique that performs a linear interpolation of inputs.

Our proposed LM instead mixes the latent representations between positive and negative pairs given by the student and teacher networks as opposed to mixing the raw images. The motivations for this is that  $f_\theta^S$  learns more about the geometry of the embedding space induced by  $f_\omega^T$  and interpolating on a lower-dimensional manifold than the original input can lead to smoother interpolations. The interpolation function  $\kappa(\mathbf{z}_i, \mathbf{z}_j)$  in Equation 7.8 outputs a contrastive mixture  $\tilde{\mathbf{z}}$  from  $\mathbf{z}_i \in \mathbb{R}^d$ ,  $\mathbf{z}_j \in \mathbb{R}^d$  and the mixture coefficient  $\nu \in [0, 1]$  is drawn from the beta distribution  $\nu \sim \text{Beta}(\beta, \beta)$  where  $\beta \in [0, \infty]$  and  $\beta \rightarrow 0$  approaches the empirical risk.

$$\kappa(\mathbf{z}_i, \mathbf{z}_j) = \nu \mathbf{z}_i + (1 - \nu) \mathbf{z}_j \quad (7.8)$$

Both student and teacher LM representations and teacher targets are then computed as,

$$\tilde{\mathbf{z}}_{ij}^S = \kappa(\mathbf{z}_i^S, \mathbf{z}_j^S), \quad \tilde{\mathbf{z}}_{ij}^T = \kappa(\mathbf{z}_i^T, \mathbf{z}_j^T) \quad (7.9)$$

$$\tilde{\mathbf{y}}_{ij}^S = \sigma(\mathbf{W}^T \tilde{\mathbf{z}}_{ij}^S / \tau), \quad \tilde{\mathbf{y}}_{ij}^T = \sigma(\kappa(\mathbf{y}_i^T, \mathbf{y}_j^T) / \tau) \quad (7.10)$$

where  $\tilde{\mathbf{y}}_{ij}^T$  is a synthetic bimodal mixup target. Henceforth, we will denote mixup teacher targets as  $\tilde{\mathbf{y}}^T$  and LM representations as  $\tilde{\mathbf{z}}^S$  and  $\tilde{\mathbf{z}}^T$ . The objective can then be described by the KLD as Equation 7.11 where  $\mathbb{H}$  is the entropy of the predicted teacher distribution over classes  $\tilde{\mathbf{y}}^T$ . When training from scratch with standard cross-entropy, the targets are mixed and renormalized with  $\sigma$  where  $\tau$  performs label smoothing resulting in a peaked bimodal distribution.

$$D_{\text{KLD}}(\tilde{\mathbf{y}}^T || \tilde{\mathbf{y}}^S) = \mathbb{H}(\tilde{\mathbf{y}}^T) - \tilde{\mathbf{y}}^T \log(\tilde{\mathbf{y}}^S) \quad (7.11)$$

Instead of using contrastive representation mixup with the KLD distillation objective, we also use it to mix between latent representations in the CL setting whereby representations of negative and positive samples are mixed to produce pseudo-*hard* negative sample representations. In this case  $\tilde{\mathbf{z}}^S := \kappa(\mathbf{z}_i^S, \mathbf{z}_j^T)$  and similarly for the teacher network as shown in Line 12 of Algorithm 2.

### 7.3.3 Theoretical Analysis of Conditioned Sampling

In this subsection we reformulate the MI lower bound to include the notion of semantic similarity between negative samples and their corre-

sponding anchor. We then describe the difference in sample complexity between USNs and SCNS w.r.t. observing the top- $k$  negative samples in the training data. By showing that the top- $k$  semantically similar negatives approximate uniform sampling of all negative we replace the respective term in the MI lower bound to formalize the objective.

We use the InfoNCE loss [321] with SCNS for our experiments, as shown in Equation 7.12,

$$\ell = \mathbb{E}_{\substack{(\mathbf{x}_*, \mathbf{x}_+) \sim \mathcal{D}_+ \\ \mathbf{x}_- \sim \mathcal{D}_- \propto P_x}} \left[ -\log \left( \frac{\exp(\mathbf{z}_*^\top \mathbf{z}_+)}{\exp(\mathbf{z}_*^\top \mathbf{z}_+) + \exp(\mathbf{z}_*^\top \mathbf{z}_-)} \right) \right] \quad (7.12)$$

where  $\mathbf{x}_-$  is conditioned on the distribution  $P_x$  as described in Section 7.3.1. By minimizing the InfoNCE loss  $\ell$ , we maximize the MI between the positive pair  $(\mathbf{x}_*, \mathbf{x}_+)$ . The optimal score for  $f(\mathbf{x}_*, \mathbf{x}_+)$  is given by  $p(x_*|x_+)/p(x_*)$ , substituting this into Equation 7.12 and splitting  $\mathbf{x}$  into positive and negative samples  $X_-$  gives:

$$\begin{aligned} \ell &= -\mathbb{E}_X \log \left[ \frac{p(x_*|x_i)/p(x_*)}{\frac{p(x_*|x_+)}{p(x_*)}} \right] \\ &+ \sum_{x_i \in X_-} \frac{p(x_i|x_+)}{p(x_i)} \Bigg] = \mathbb{E}_X \log \left[ 1 + \frac{p(x_*)}{p(x_*|x_+)} + \sum_{x_i \in X_-} \frac{p(x_i|x_+)}{p(x_i)} \right] \\ &\approx \mathbb{E}_X \log \left[ 1 + \frac{p(x_*)}{p(x_*|x_+)} (M-1) \mathbb{E}_{x_i} \frac{p(x_i|x_+)}{p(x_i)} \right] \\ &= \mathbb{E}_X \log \left[ 1 + \frac{p(x_*)}{p(x_*|x_+)} (M-1) \right] \geq \mathbb{E}_X \log \left[ \frac{p(x_*)}{p(x_*|x_+)} M \right] \\ &= -I(x_*, x_+) + \log(M) \end{aligned} \quad (7.13)$$

From Equation 7.13, we see that  $I(x_*, x_+) \geq \log(M) - \ell$  [321] and the larger the number of negative samples,  $M$ , the tighter the MI bound. However, we argue that if a pretrained  $f_\omega^T$  has training error close to 0, then  $\log(M)$  should be replaced with a term that accounts for the geometry of the embedding space as not all negative samples are equally important for reducing  $\ell$ . Therefore, we express how top- $k$  samples from SCNS tightens the lower bound estimate on MI when compared to using USNs. Given that we are not restricted to a distance, divergence or similarity between vectors, we refer to a general *alignment* function  $A$  that outputs an alignment score  $a \in [0, 1]$ .

Given  $\mathbf{x}_*$ , the expected alignment score for the top- $k$  negative samples is  $a_{x_-}^k := \mathbb{E}_{x_- \sim D_x^k} [A(z_*, z_-)]$  and for negative samples outside of the top- $k$  samples,  $a_{x_-}^r := \mathbb{E}_{x_- \sim D_x^r} [A(z_*, z_-)]$  where  $D_{x_*}^r \subseteq D, D_{x_*}^k \notin D_{x_*}^r$ ,  $r = N - N_y - k$ . and  $N_y$  is the number of samples of class  $y$ . The alignment weight (AW)  $\Omega_x := 1 - a_x^k / (a_x^k + a_x^r)$  is then used to represent the difference in ‘closeness’ between the top- $k$  negative samples and the remaining negative samples.

**Lemma 1.** Given  $\Omega := \sum_{i=1}^M \Omega_{x_*}$ , we can reformulate the MI lower bound when using SCNS as Equation 7.14.

$$I(X, Y) \geq \ell + \log(2\Omega) \quad (7.14)$$

*Proof.* We substitute  $\log(2\Omega)$  for  $\log(M)$  in Equation 7.13 as  $a_x^k \approx a_x^r$  in uniform sampling as  $M \rightarrow \infty$ .  $\square$

This lower bound favors top- $k$  negative samples that have alignment with the positive class boundary and are relatively close compared to the negative samples outside of the top- $k$ . It is dependent on the loss of  $f^*$  where  $\ell \approx 0$  results in an accurate alignment estimation for all embedding pairs.

**Lemma 2.** In the worst case, when all  $M$  negatives are equidistant to  $x_*$ , forming a ring on the  $L_2$  embedding hypersphere, SCNS is equivalent to uniform sampling.

*Proof.* This holds as  $\log(2\Omega) \approx \log(M)$  when the centroids of both sets  $a_x^k = a_x^r$  and  $\Omega = M/2$ . Therefore, in the worst case SCNS is equivalent to uniform sampling.  $\square$

The above case can be due to degenerative representations used in the top- $k$  SCNS similarity computation (i.e  $\epsilon_{tr}$  not close to 0) or a characteristic of the training data itself. Then, the relation between  $M$  USNs and top- $k$  SCNSs can be formed as follows. Let  $\Omega_{D_x^u}$  be the AW of USNs for  $x$  and  $\Omega_{D_x^k}$  be the AW for the top- $k$  negative samples. When  $|D_x^u| \approx |D_x^k|$  we have,

$$I(X, Y) \geq \ell + \log(2\Omega_{D_x^k}) \geq \ell + \log(2\Omega_U) \quad (7.15)$$

given the non-uniform prior over negative samples as defined in SCNS. For some  $k \ll N$ ,  $2(\Omega_k - \Omega_U) = 0$  is met when a subset of  $D_x^u$  negative samples have  $\bar{z}_{\tilde{x}}^u \approx \bar{z}_x^k$  where  $\tilde{x}$  denotes the aforementioned subset and  $\bar{z}_x^k$  is the centroid of the top- $k$  negative samples.

NUMBER OF UNIFORM DRAWS TO OBSERVE TOP- $k$  SCNS We can now describe the expected number of USNs draws required to observe the top- $k$  samples at least once for a given  $x_*$ . Let  $C_i$  denote the number of negative samples observed until the  $i$ -th new negative sample among the top- $k$  samples is observed and  $N$  is the total number of samples until all top- $k$  negative samples are observed. Since  $C = \sum_{i=1}^k C_i$ ,  $\mathbb{E}[C] = \mathbb{E}\left[\sum_{i=1}^k C_i\right] = \sum_{i=1}^k \mathbb{E}[C_i]$  where  $C_i$  follows a geometric distribution with parameter  $(k+1-i)/M$ . Therefore,  $\mathbb{E}[C_i] = M/(k+1-i)$  and thus the expected number of draws is given by Equation 7.16.

$$\mathbb{E}[C] = M \sum_{i=1}^k (k+1-i)^{-1} = M \sum_{i=1}^k i^{-1} \quad (7.16)$$

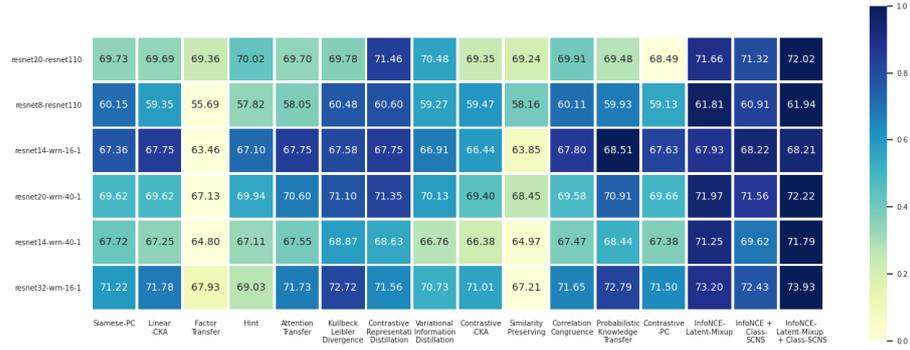


Figure 7.2: CIFAR-100 Test Accuracy for Knowledge Distillation Approaches

We reformulate this for mini-batch training where consecutive batches of size  $b$  for  $x_*$  are drawn with replacement. This is a special case of the *Coupon Collector’s Problem* [(CCP) 430].

**Theorem 3.** *The batch variant of the CCP formulates the probability of the expected number of batches of size  $b$  to observe top- $k$  SCNS samples at least once as Equation 7.17*

$$\sum_{j=0}^{\infty} P(K > ib) = \sum_{j=0}^{M-1} (-1)^{M-j+1} \binom{M}{j} \frac{1}{1 - \left(\frac{j}{M}\right)^b} \quad (7.17)$$

*Proof.* See Section A.2.6 for the proof.  $\square$

As  $M$  grows more mini-batches are required to observe the top- $k$  hard negative samples. Thus,  $b$  has to be larger for uniform sampling to cover the informative negative samples, coinciding with the MI formulation in Equation 7.15. Further justifications are found in Section A.2.5 and Section A.2.6.

## 7.4 EXPERIMENTS

We now discuss the experimental results and note that additional details on hardware, datasets, model architectures and settings are found in Section A.2.1, A.2.2 and A.2.3.

**STANDARD SUPERVISED LEARNING** We first test our three NS approaches in the standard supervised learning setting on CIFAR-10. Several ResNet-based architectures [151, 455, 480] are trained with (1) cross entropy between LM representations and the cross-entropy between student predictions and targets (Cross-Entropy + LM) and CL (InfoNCE + LM), (2) only using CL with the LM representations of each contrastive pair (InfoNCE-LM) and (3) Class-SCNS (InfoNCE + Class-SCNS) and Instance-SCNS (InfoNCE + Instance-SCNS) with the InfoNCE loss.

Table 7.1 shows this ablation, where bolded results represent the best performance within the horizontal lines of that section and  $\dagger\dagger$  corresponds

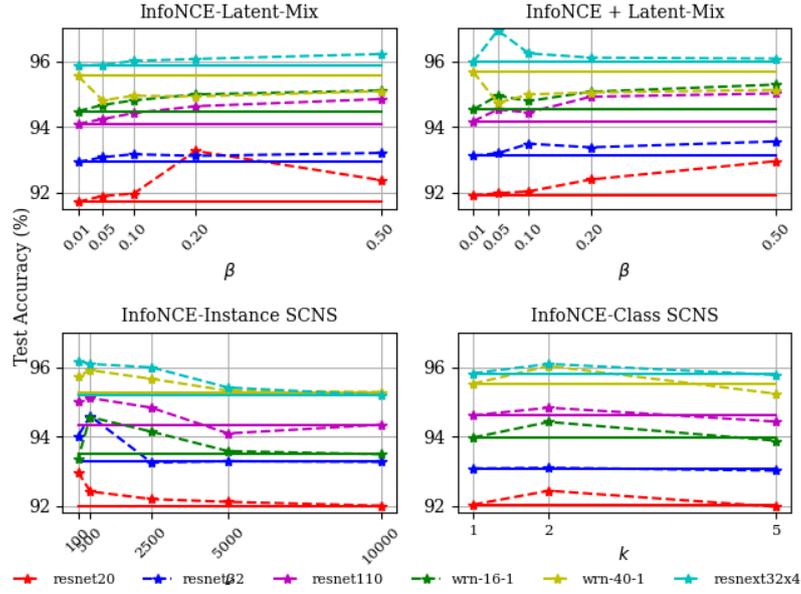
Methods	resnet20	resnet32	resnet110	wrn-16-1	wrn-40-1	resnext32x4
Cross-Entropy	91.14	92.49	93.38	94.06	94.47	95.38
Cross-Entropy + LM ( $\beta=0.5$ )	92.74	92.73	93.53	94.26	94.54	95.47
InfoNCE	91.68	92.90	94.01	94.39	95.23	95.77
-LM ( $\beta=0.01$ )	91.72	92.93	94.09	94.48	95.55	95.87
-LM ( $\beta=0.05$ )	91.90	93.08	94.24	94.67	94.81	95.89
-LM ( $\beta=0.1$ )	91.97	93.17	94.42	94.81	94.95	96.01
-LM ( $\beta=0.2$ )	92.17	93.12	94.63	94.99	94.93	96.07
-LM ( $\beta= 0.5$ )	<b>92.38</b>	<b>93.21</b>	<b>94.85</b>	<b>95.11</b>	<b>95.09</b>	<b>96.22</b>
+ LM ( $\beta=0.01$ )	91.92	93.12	94.17	94.53	95.68	95.97
+ LM ( $\beta=0.05$ )	91.98	93.20	94.53	94.95	94.73	<b>96.94</b>
+ LM ( $\beta=0.1$ )	92.04	93.49	94.44	94.79	94.99	96.24
+ LM ( $\beta=0.2$ )	92.40	93.38	94.92	95.07	95.05	96.11
+ LM ( $\beta=0.5$ )	<b>92.96</b> <sup>††</sup>	93.56	<b>95.02</b>	<b>95.29</b> <sup>††</sup>	<b>95.13</b>	96.08
+ Class-SCNS ( $k = 1$ )	92.04	93.08	94.61	93.97	95.52	95.82
+ Class-SCNS ( $k = 2$ )	<b>92.44</b>	93.10	94.83	94.42	<b>96.03</b> <sup>††</sup>	<b>96.09</b>
+ Class-SCNS ( $k = 5$ )	91.98	93.02	<b>94.43</b>	<b>93.88</b>	95.23	95.77
+ Instance-SCNS ( $k= D /5$ )	92.01	93.27	94.34	93.49	95.28	95.19
+ Instance-SCNS ( $k= D /10$ )	92.12	93.29	94.09	93.58	95.32	95.41
+ Instance-SCNS ( $k= D /20$ )	92.20	93.25	94.83	94.14	95.66	95.99
+ Instance-SCNS ( $k= D /100$ )	<b>92.42</b>	<b>94.39</b> <sup>††</sup>	<b>95.11</b> <sup>††</sup>	<b>94.56</b>	<b>95.91</b>	96.09
-Instance SCNS ( $k= D /500$ )	92.38	93.57	95.02	93.36	95.72	<b>96.27</b> <sup>††</sup>

Table 7.1: Test accuracy (%) of student networks on CIFAR-10.

to the best performance overall for the respective architecture. For ‘InfoNCE-LM’ and ‘InfoNCE + LM’,  $\beta = 0.01$  corresponds to a slight mixing of negative pair latent representations and  $\beta = 0.5$  leads to a U-shape probability density function in  $[0, 1]$ , leading to increased LM.

We find that using the original negative samples and the latent mixture features (InfoNCE + LM) improves over only using the LM features (InfoNCE-LM). We also find that using LM with label smoothing improves cross-entropy training over only using cross-entropy training and that increased LM ( $\beta = 0.5$ ) improves performance for both InfoNCE + LM and InfoNCE-LM. Hence, this suggests LM performs well for both point-wise and pairwise based supervised training. We note that the regularization term for the LM loss is manually searched over settings  $\alpha \in [0.01, 0.05, 0.1, 0.2]$  on a validation dataset which is 5% randomly sampled from the predefined CIFAR-10 training data. The results reported in Table 7.1 are with a regularization term set  $\alpha = 0.1$ . Figure 7.3 visualizes the effect of changing  $\beta$  for LM and  $k$  for Class-SCNS and Instance-SCNS for 5 different ResNet-based architectures. We find that on average InfoNCE + LM models outperforms InfoNCE-LM that setting  $\beta \approx 0.5$ . For Instance-SCNS, we find that  $k = 500$  for Wide-ResNet architectures and  $k = 100$  for ResNet architectures leads to increases in test accuracy.

KNOWLEDGE DISTILLATION We now discuss results of the KD experiments on CIFAR-100 [209]. Figure 7.2 shows the test accuracies for each KD method where the student-teacher pair is different for each row and the colours correspond to  $[0-1]$  row normalized test accuracies to visualize the relative percentage increase or decrease between

Figure 7.3: Effect of  $\beta$  in Latent Mixup and  $k$ in SCNS

each  $KD$  method<sup>1</sup> The y-axis model naming convention is (student-# convolutional layers-teacher-# convolutional layers-# fully-connected-layers) and the x-axis denotes the  $KD$  method.

Teacher Network	resnet152	resnet152	resnet50	resnet50	resnet152	densenet121	resnet50	resnet50	resnet34	resnet50
Student Network	resnet56	resnet44	resnet20	resnet32	resnet32	resnet32	resnet8	resnet14	resnet14	resnet32x4
KLD [102]	52.22	51.29	47.80	50.69	50.90	48.61	42.05	44.91	45.63	62.41 <sup>†</sup>
AT [178]	43.81 (-8.41)	47.45 (-3.84)	45.48 (-2.32)	47.47 (-3.89)	47.01 (-2.22)	46.39 (-1.82)	39.45 (-2.60)	44.56 (-0.35)	45.20 (-0.43)	54.49 (-7.95)
SR [119]	48.12 (-4.10)	47.29 (-4.05)	44.30 (-3.50)	45.30 (-4.45)	46.45 (-2.26)	46.35 (-1.86)	38.47 (-3.57)	43.51 (-1.39)	44.08 (-1.55)	53.07 (-9.37)
PKT [330]	49.14 (-3.08)	48.82 (-2.47)	46.42 (-1.38)	48.03 (-3.21)	47.69 (-0.39)	48.22 (+0.01)	40.10 (-1.85)	45.18 (+0.27)	45.05 (-0.58)	55.00 (-7.43)
CC [333]	42.35 (-9.87)	45.27 (-6.02)	46.59 (-1.21)	46.56 (-4.15)	46.75 (-1.41)	47.20 (-1.01)	40.08 (-1.97)	45.02 (+0.11)	44.10 (-1.53)	49.06 (-13.37)
VID [4]	48.49 (-3.73)	48.12 (-3.17)	45.59 (-2.21)	47.63 (-2.81)	48.09 (-4.98)	43.63 (-4.57)	40.65 (-1.40)	44.40 (+0.51)	45.45 (-0.18)	54.75 (-7.69)
FT [198]	44.19 (8.03)	47.59 (-3.70)	46.03 (-1.77)	45.28 (-3.72)	47.18 (-9.57)	39.04 (-9.17)	39.67 (-2.38)	38.48 (-6.43)	44.25 (-1.38)	52.31 (-10.12)
CRD [414]	50.42 (-1.79)	47.98 (-3.31)	46.92 (-0.88)	49.15 (-1.54)	49.35 (-1.75)	49.21 (+0.60)	39.86 (-2.18)	44.34 (-0.57)	45.28 (-0.35)	56.91 (-5.53)
Stance-PC	51.99 (-0.23)	47.11 (-4.18)	45.64 (-2.16)	46.42 (3.99)	46.91 (-1.81)	46.80 (-1.41)	38.03 (-4.02)	43.83 (-1.08)	44.59 (-1.04)	52.23 (-10.21)
Contrastive-PC	45.18 (-7.03)	47.19 (-4.10)	45.52 (-2.28)	46.64 (+0.01)	50.89 (-1.40)	47.21 (-1.0)	39.40 (-2.65)	44.91 (0.0)	44.09 (-1.54)	54.22 (-8.21)
Contrastive-CKA (RBF)	47.57 (-4.65)	48.71 (-2.58)	46.29 (-1.51)	46.87 (-3.59)	47.31 (-1.54)	47.07 (-1.14)	41.28 (-0.77)	47.05 (+2.14)	44.76 (-0.87)	52.50 (-9.93)
Contrastive-CKA (Linear)	47.87 (-4.35)	47.90 (-3.39)	45.89 (-1.91)	47.42 (-3.27)	41.13 (-1.62)	46.99 (-1.21)	40.03 (-2.02)	44.63 (-0.28)	44.51 (-1.12)	54.16 (-8.28)
Our										
CRD-LM	52.00 (- 0.22)	50.28 (-1.01)	48.12 (+0.32)	50.23 (-1.22)	49.68 (+1.10)	49.71 (+1.50)	42.18 (+0.13)	46.08 (+1.17)	45.73 (+0.09)	58.53 (-3.91)
InfoNCE + Class-SCNS	53.03 (+0.61)	49.89 (-1.40)	47.02 (-0.78)	49.80 (-1.28)	49.62 (+ 0.60)	49.22 (+0.61)	41.52 (-0.53)	45.39 (+0.48)	45.12 (-0.51)	58.36 (-4.08)
InfoNCE + Instance-SCNS	52.24 (+ 0.02)	51.13 (-0.16)	47.44 (-0.63)	50.04 (-.75)	50.27 (+1.77) <sup>††</sup>	50.38 (+1.77)	41.29 (-0.76)	46.12 (+1.21)	45.80 (+0.17)	58.95 (-3.49)
InfoNCE-LM + Class-SCNS	52.83 (+ 0.61)	51.33 (+0.04)	48.72 (+0.92)	50.83 (+0.75)	50.15 (+1.29)	49.90 (+1.69)	43.44 (+1.39)	46.99 (+2.08)	46.01 (+0.38)	59.25 (-3.19)
InfoNCE-LM + Instance-SCNS	53.21 (+ 0.99) <sup>††</sup>	52.72 (+1.43) <sup>††</sup>	49.07 (+1.27) <sup>††</sup>	51.18 (+0.48)	50.42 (+1.50)	50.11 (+1.89) <sup>††</sup>	43.89 (+1.84) <sup>††</sup>	47.32 (+2.41) <sup>††</sup>	46.83 (+1.20) <sup>††</sup>	60.09 (-2.35)

Table 7.2: Test accuracy (%) of student networks on Tiny-ImageNet-200

We find that combining LM with InfoNCE + Instance-SCNS with a lookup table outperforms only using LM or instance-level SCNS with a lookup table. Moreover, ‘InfoNCE-LM + Class-SCNS’ outperforms all other  $KD$  methods for all but one student-teacher pairing (PKT found to have the highest test accuracy for ‘resnet14-wrn-16-1’ student-teacher pair). However, the original KLD distillation loss remains a very strong baseline that is competitive with CL and even outperforms our proposed non-contrastive baselines. We find a 0.24 point increase in the 0-1 normalized average score (‘Kullback Leibler Distillation’ = 0.75 and ‘InfoNCE-LM + Class-SCNS’=0.99). We also find w.r.t. the student-teacher network capacity gap, increasing the capacity of the teacher network does not necessarily lead to improved student

<sup>1</sup> The naming conventions of baselines, including *our own*  $KD$  baselines are described in Section A.2.2.

network performance if the gap is large. The performance difference between ‘resnet14-wrn-16-1’ and ‘resnet14-wrn-40-1’ is relatively small and ‘resnet14-wrn-40-1’ has higher accuracy than ‘resnet14-wrn-16-1’ in only 7/16 different loss functions. However, in 3/4 of the CL cases (4 rightmost columns of Figure 7.2) the larger teacher network in ‘resnet14-wrn-40-1’ has significantly improved accuracy.

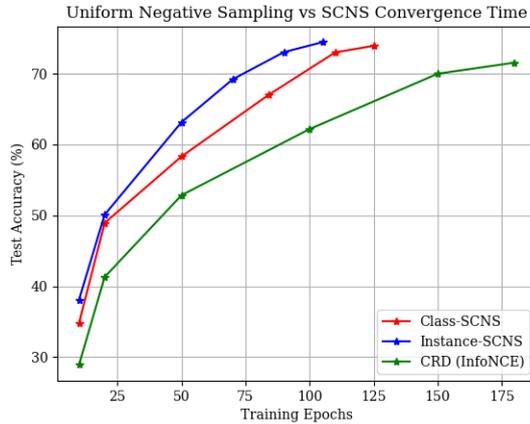


Figure 7.4: CIFAR-100 Convergence Time Comparison

Figure 7.4 shows the convergence time comparing InfoNCE when using USNs and SCNS for the resnet32-wrn-16-1 student-teacher pair on CIFAR-100. We see that Instance-SCNS converges after 106 training epochs, Class-level SCNS at 124 epochs while USNs converges at 181 epochs. Hence, both test accuracy and convergence time is improved by sampling hard negative samples via SCNS.

**DISTILLING TRANSFER LEARNED REPRESENTATIONS** To test how SCNS performs in the transfer learning setting, we learn a student network from teacher network that takes inputs and outputs of different sizes. We use pretrained ImageNet models and fine-tune them on Tiny-ImageNet-200 by replacing the last 1000 dimensional linear layer with a 200 dimensional layer. The pretrained models are fine-tuned by resizing Tiny-ImageNet-200 images from 64x64 to 256x256 without any additional data augmentation. These models are used as the teacher networks that take in 256x256 images while the student network takes the original 64x64 input. This KD setup is slightly different as the teacher network is fine-tuned using transfer learning from the original ImageNet dataset, not from random initialization.

Table 7.2 shows each KD technique along with our proposed techniques from row ‘Contrastive-PC’ to the last row. In almost all student-teacher network combinations, Instance-SCNS, Class-SCNS samples and both with Contrastive LM regularization have led to performance improvements over all previously proposed KD methods. However, the original KLD distillation loss [162] remains a very strong baseline. We also

Teacher Network	resnet110	resnet110	wrn-16-1	wrn-16-1	wrn-40-1	wrn-40-1
Student Network	resnet20	resnet8	resnet14	resnet32	resnet20	resnet14
Teacher (Cross-Entropy)	74.31	74.31	73.11	73.40	75.43	75.43
Student (Cross-Entropy)	66.26	57.31	65.84	69.80	67.69	68.02
Student (Cross-Entropy + KLD)	69.78	60.48	67.58	71.10	68.87	72.72
InfoNCE						
-LM ( $\beta = 0.01$ )	68.13	58.32	67.11	70.09	69.02	68.55
-LM ( $\beta = 0.05$ )	68.15	58.41	67.25	69.98	69.27	68.80
-LM ( $\beta = 0.1$ )	68.22	58.89	67.48	70.49	69.49	69.39
-LM ( $\beta = 0.2$ )	68.29	<b>59.13</b>	67.59	<b>71.42</b>	69.88	69.97
-LM ( $\beta = 0.5$ )	<b>68.49</b>	59.07	<b>67.63</b>	69.66	<b>69.91</b>	<b>70.90</b>
+LM ( $\beta = 0.01$ )	68.71	58.31	67.23	70.33	68.92	68.52
+LM ( $\beta = 0.05$ )	68.83	58.78	67.61	70.57	69.10	68.80
+LM ( $\beta = 0.1$ )	68.96	59.01	68.45	<b>70.92</b>	69.48	69.04
+LM ( $\beta = 0.2$ )	69.08	<b>61.09</b>	69.58	70.83	71.03	70.24
+LM ( $\beta = 0.5$ )	<b>69.25</b>	60.70	<b>70.09</b>	69.66	<b>71.29</b>	<b>71.43</b>
-Class SCNS ( $k =  V /100$ )	70.03	63.20	68.48	69.31	69.44	70.41
-Class SCNS ( $k =  V /50$ )	70.88	<b>64.03</b>	<b>69.73</b>	71.09	69.62	<b>72.83</b>
-Class SCNS ( $k =  V /20$ )	<b>71.32</b> <sup>††</sup>	63.91	69.36	71.56	<b>71.37</b> <sup>††</sup>	72.21
-Class SCNS ( $k =  V /10$ )	71.03	62.14	68.73	<b>72.02</b>	71.02	72.08
-Class SCNS ( $k =  V /5$ )	69.35	61.51	68.05	70.99	69.81	71.76
-Instance SCNS ( $k =  D /100$ )	70.24	63.69	68.92	70.01	69.75	70.20
-Instance SCNS ( $k =  D /50$ )	71.21	64.99	69.23	<b>72.94</b> <sup>††</sup>	69.70	<b>73.10</b> <sup>††</sup>
-Instance SCNS ( $k =  D /20$ )	<b>71.25</b>	<b>65.13</b> <sup>††</sup>	<b>70.11</b> <sup>††</sup>	71.87	69.31	72.09
-Instance SCNS ( $k =  D /10$ )	71.04	64.87	70.03	71.83	<b>70.14</b>	72.02
-Instance SCNS ( $k =  D /5$ )	70.93	64.84	69.82	70.53	69.31	71.80

Table 7.3: An ablation of efficient NS techniques on CIFAR-100.

find that increasing the capacity of the teacher network for the same sized student network can result in the same or poorer performance if the original student-teacher network capacity difference is large. For example, if we compare ‘-resnet14-wrn-16-1’ to ‘resnet14-wrn-40-1’ we can see there is little difference in performance across the different KD methods. However, increasing the student network size closer to the teacher network leads to improved performance e.g ‘resnet32-wrn-16-1’ consistently improves over ‘resnet14-wrn-16-1’.

Table 7.3 shows the ablation of all three proposed methods on CIFAR-100. The most consistent gain in performance is found when using Instance-SCNS as it achieves the best performance for 4 out of 6 student networks. Class-SCNS performs the best for resnet20 student networks, which have relatively larger capacity compared to resnet8 and resnet14.

#### 7.4.1 Summary

We proposed (1) semantically conditional negative sampling, a method that use pretrained networks to define a negative sampling distribution and (2) latent mixup, a simple strategy to form *hard* negative samples. We found that when used in a contrastive learning setting, both proposals consistently outperform previous knowledge distillation methods and

improve contrastive learned models in the standard supervised learning setup.



# 8

---

## MODEL COMPRESSION VIA LAYER FUSION

---

In this chapter, we describe *layer fusion*, a novel model compression technique that fuses the weight information of similar fully-connected, convolutional or attention layers. Models that use layer fusion can significantly reduce the number of layers of the original network while maintaining competitive performance on real-world problems, such as image classification and language modelling. We also find for very deep network architectures, other well-established compression techniques can achieve competitive performance to their original networks given a sufficient number of retraining steps. From experiments on CIFAR-10, we find that various deep convolution neural networks can remain within 2% accuracy points of the original networks up to a compression ratio of 3.33 when iteratively retrained using layer fusion. For experiments on the WikiText-2 language modelling dataset where transformer models are used, we achieve compression that leads to a network that is 20% of its original size while being within 5 perplexity points of the original network. For both tasks, there is a clear inflection point observed in performance as the amount of compression increases, suggesting a bound on the amount of compression before an exponential degradation in performance begins.

### 8.1 INTRODUCTION

DNNs have made a significant impact on fields such as CV [151, 178] and NLP [90, 424]. Deep CNNs [210] have improved performance on image classification [210], image segmentation [261], speech recognition [221] and have been widely adopted in the machine learning (ML) community. This has been accelerated due to numerous innovations such as skip connections in ResNets [151] to avoid the vanishing gradient problem and batch normalization [18, 180] and layer normalization [18] to reduce the effects of shifts in the training and test data distributions. Similarly, Transformer models have shown great success in NLP due to the use of self-attention [424], significantly outperforming preceding RNN based architectures [165] on a diverse set of NLP tasks [80, 84, 349]. However, these large overparameterized networks require more compute, training time, storage and leave a larger carbon footprint [399]. While prior work on model compression has mainly focused on deploying compressed

models to mobile devices [144, 447], moving models from multi-GPU training to single-GPU training is now too a salient challenge. If achieved, this relaxes the resource requirements for ML practitioners and allow a wider adoption of larger pretrained CNNs and Transformers.

This leads us to ask the following questions on DNNs: *are all layers of large pretrained models required for a given target task? If not, can we reduce the network while preserving network density (i.e no non-zero parameters) during retraining in a computationally efficient manner?* Earlier work [151] on CNNs found that some layers may become redundant in very deep networks, essentially copying earlier layers and performing identity mappings for the redundant layers. While residual connections have ameliorated these problems to some degree (not only in residual networks e.g Transformers), we assert that there may still be significant redundancy between layers of large overparameterized networks. More recently, Zhang, Bengio, and Singer [482] found that whole layers can be distinctly separated by their importance in prediction, further motivating us to seek a compression technique that identifies and uses salient layers. However, many current compression techniques are not equipped to preserve these salient layers during model compression of pretrained models because they are unstructured techniques [145, 147, 191], resulting in a sparse model. This is a practical limitation since sparse networks require more conditional operations to represent which elements within each parameter matrix are zero or non-zero. Current GPU libraries such as CuSPARSE (accounting for recent improvements [12]) are far slower than CuBLAS [370] and current hardware is not designed to optimize for sparse matrix operations.

In contrast, knowledge distillation [17, 162, 287] and quantization [346] preserve network density, avoiding the necessity for specialized sparse matrix libraries to utilize the benefits of smaller and faster networks. However, quantization leads to quantization error and requires approximate methods to compute partial derivatives during retraining [3] and knowledge distillation requires more memory to store and train the smaller network. Weight sharing reduces the network size and avoids sparsity, however it is unclear which weights should be shared and it cannot be used when the model is already pretrained with untied weights. The noted drawbacks of the aforementioned compression methods further motivates us to seek an alternative structured compression method that preserves network density while identifying and removing redundant layers. This brings us to our main contributions.

**CONTRIBUTIONS** We propose *layer fusion* (LF). LF aims to preserve information across layers during retraining of already learned models while preserving layer density for computational efficiency. We also propose *alignment* measures for LF, since aligning paths, layers and whole neural networks is non-trivial (neurons can be permuted and still exhibit similar behaviour and we desire an invariance to orthogo-

nal transformations). This includes (1) a Wasserstein distance metric to approximate the alignment cost between weight matrices and (2) numerous criteria for measuring similarity between weight covariance matrices. We use these measures as LF criteria to rank layer pairs that are subsequently used to fuse convolutional, fully-connected and attention layers. This leads to both computational and performance improvements over layer removal, pruning and shows competitive results compared to tensor-decomposition and unsupervised knowledge distillation. We report results of LF using different fusion approaches: layer freezing, averaging and random *mixing*. We use current structured compression techniques as baselines for both CNNs and Transformers, with and without retraining and identify a reoccurring inflection point w.r.t performance versus model size.

## 8.2 RELATED WORK

**LAYER STRUCTURE & IMPORTANCE** Zhang, Bengio, and Singer [482] have recently analysed the layer-wise functional structure of over-parameterized deep models to gain insight into why deep networks have performance advantages over their shallow counterparts. They find that some layers are salient and that once removed, or reinitialized, have a catastrophic effect on learning during training and subsequently generalization. In contrast, the remaining layers once reset to their default initialization has little effect. This suggests that parameter and norm counting is too broad of a measure to succinctly study the generalization properties in deep networks. These findings also motivate LF, as we posit that important layers are more distinct and therefore will be less similar, or harder to align with other layers, while more redundant layers may be good candidates for fusing layers. Frankle and Carbin [111] showed that there exists trained subnetworks that when re-initialized to their original configuration produce the same performance as the original network in the same number of training epochs. They also posit that stochastic gradient descent (SGD) seeks out a set of *lottery tickets* (i.e well-initialized weights that make up a subnetwork that when trained for the same number of epochs as the original network, or less, can reach the same out-of-sample performance) and essentially ignores the remaining weights. We can further conjecture from Zhang, Bengio, and Singer [482] findings, that perhaps SGD more generally seeks out important layers, which we analogously refer to as *lottery pools*. Identifying whole layers that are significantly distinguished from others, in terms of their influence on learning, further motivates us to merge or freeze layers.

**COMPUTING LAYER SIMILARITY** Kornblith et al. [206] have focused on computing similarity between different neural representations (i.e the activation output vector for a given layer). However, we view

this comparison between layers as slightly limiting, since information is lost about what weights and bias led to the activation outputs. Moreover, directly comparing neural network weights allows us to avoid sampling inputs to compute the activations. In contrast, work focusing on representational similarity across networks Kornblith et al. [206] and Li et al. [236], we are instead comparing weight matrices within the same network. Directly comparing weights and biases allow us to better approximate alignments and similarities for dense networks and has the advantage that we do not require data to be feed-forward through the network post-training to measure similarity within or across networks, unlike representational similarity (i.e output activations).

Structured Dropout [104] proposes to randomly drop whole layers during training time and at test time they can choose a subnetwork which can be decided based on performance of different combinations of pruned networks on the validation set or based on dropout probabilities learned for each layer throughout training. Singh and Jaggi [394] measure model similarity across neural networks using optimal transport-based metrics. In contrast, our work measures intra-network similarity and we make a distributional assumption that allows us to use such metric efficiently during retraining, making it feasible to scale for large networks.

### 8.3 METHODOLOGY

We begin by defining a dataset as  $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \dots, T\}$  that contains  $T$  tuples of an input vector  $\mathbf{x} \in \mathbb{R}^n$  and a corresponding one-hot target  $\mathbf{y} \in \{0, 1\}^p$  where  $p$  is the dimensionality. We define any arbitrary sample as  $s := (\mathbf{x}, \mathbf{y})$  where  $s \in \mathcal{D}$ . We consider a neural network  $f_\theta(\mathbf{x})$  with pretrained parameters  $\theta := (\theta_1, \theta_2, \dots, \theta_\ell, \dots, \theta_L)^T$ . Here  $\theta_\ell := \{\mathbf{W}_\ell, \mathbf{b}_\ell\}$  where  $\mathbf{W}_\ell \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$ ,  $\mathbf{b}_\ell \in \mathbb{R}^{n_\ell}$  where  $n_\ell$  denotes the dimension size of the  $\ell$ -th layer. Thus, a standard fully-connected  $f_\theta$  is expressed as,

$$f_\theta(\mathbf{x}) := \mathbf{W}_L g \left( \dots g \left( \mathbf{W}_2 g \left( \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1 \right) + \mathbf{b}_2 \right) + \mathbf{b}_L \right) \quad (8.1)$$

with smooth asymptotic nonlinear function  $g(\cdot)$  (e.g hyperbolic tangent) that performs elementwise operations on its input. The input to each subsequent layer as  $\mathbf{z}_\ell \in \mathbb{R}^{n_\ell}$  where  $\mathbf{x} := \mathbf{z}_0$  for  $m$  number of units in layer  $\ell$  and the corresponding output activation as  $\mathbf{T}_\ell = g(\mathbf{z}_\ell)$ . The loss function is defined as  $\mathbb{L}_\theta(\mathcal{D}) := \frac{1}{T} \sum_{i=1}^T \mathcal{L}(\mathbf{y}_i, f_\theta(\mathbf{x}_i))$  where for a single sample  $s_i$ ,  $\mathcal{L} : \mathcal{Y} \times \mathbb{R}^n \rightarrow \mathbb{R}$ . A pruned  $\theta_\ell$  post-training is denoted as  $\theta_\ell^p$  and a tensor decomposed  $\theta_\ell$  is expressed as  $\tilde{\theta}_\ell$  where  $\tilde{\mathbf{W}}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$  and  $\tilde{\mathbf{b}}_\ell \in \mathbb{R}^{d_{\ell+1}}$  and  $d \ll n$ . A network pruned by layer as a percentage of the lowest weight magnitudes is denoted as  $f_\theta^{lp}$  where the pruned weights  $\theta' \subset \theta$ . A percentage of the network pruned by weight magnitudes across the whole network is denoted as  $f_\theta^{gp}$  (i.e global pruning). Lastly, a post layer fused network  $f_\Theta$  has fused parameters  $\Theta$ .

### 8.3.1 Desirable Properties of Weight Similarity

Ideally, we seek a measure that can compare weight matrices that are permutable and of varying length. Formally, the main challenges with aligning weight matrices  $\mathbf{W} := \{\mathbf{W}_0, \dots, \mathbf{W}_\ell, \dots, \mathbf{W}_L\}$  of different layers is that, when vectorized as  $\text{vec}(\mathbf{W}_\ell) \in \mathbb{R}^{n_\ell(n_{\ell+1})}$ ,  $\mathbf{W}_\ell$  can be permuted and still exhibit the same behavior at the output. Hence, if  $|\mathbf{W}_\ell| \neq |\mathbf{W}_{\ell+1}|$ , the measure  $S$  must allow for multisets of different lengths and permutations. Invariance to rotations, reflections and scaling are all desirable properties we aim to incorporate into measuring similarity between weight matrices. However, invariance to linear transformations has issues when there are more parameters in a layer than training samples, as pointed out by Kornblith et al. [206]. Even though our work mainly focuses on large pretrained models, we also seek a LF measure that is invariant to orthogonal transformations to overcome the aforementioned issues i.e for a similarity function  $s(\cdot, \cdot)$ ,  $s(\mathbf{W}_i, \mathbf{W}_j) = s(\mathbf{W}_i \mathbf{U}, \mathbf{W}_j \mathbf{V})$  for full-rank orthonormal matrices  $\mathbf{U}$  and  $\mathbf{V}$  such that  $\mathbf{U}^T \mathbf{U} = \mathbf{I}$  and  $\mathbf{V}^T \mathbf{V} = \mathbf{I}$ . More importantly, invariance to orthogonal transformation relates to permutation invariance [322] which is a property we account for when measuring the alignment between weight matrices. Lastly, we note that if two weight sets are of unequal size we randomly downsample the larger weight set to match the paired weight set. This is required for aligning filters in CNN networks after vectorization. We now describe a set of measures we consider for aligning and measuring the similarity of layers.

### 8.3.2 Layer Alignment & Layer Similarity

**COVARIANCE ALIGNMENT** The first Layer Fusion (**LF**) measure we consider is covariance alignment (CA). CA accounts for correlated intra-variant distances between layers, which can indicate some redundancy, although their overall distributions may differ and therefore may be good candidates for LF. Hence, we consider the Frobenius norm (denoted as subscript  $F$ ) between pairs of weight covariance matrices  $\Sigma_{\mathbf{W}_1}, \Sigma_{\mathbf{W}_2}$  and expectation  $\mathbb{E}[\mathbf{W}_1] = \mathbb{E}[\mathbf{W}_2] = 0$ . This forms a Riemannian manifold of non-positive curvature over the weight covariances. We first consider the cosine distance as the distance measures between parameter covariance matrices as Equation 8.2, where  $\|\Sigma_{\mathbf{W}}\|_F = [\text{tr}(\Sigma_{\mathbf{W}}^T \Sigma_{\mathbf{W}})]^{1/2}$ .

$$D_{\text{cos}}(\Sigma_{\mathbf{W}_1}, \Sigma_{\mathbf{W}_2}) = \frac{\text{tr}(\Sigma_{\mathbf{W}_1} \cdot \Sigma_{\mathbf{W}_2})}{\|\Sigma_{\mathbf{W}_1}\|_F \|\Sigma_{\mathbf{W}_2}\|_F} \quad (8.2)$$

If we assume both weight matrices are drawn from a normal distribution  $\mathbf{W}_1 \sim \mathcal{N}(\mu, \sigma_1)$ ,  $\mathbf{W}_2 \sim \mathcal{N}(\mu, \sigma_2)$  with identical means  $\mu = \mu_{\mathbf{W}_1} = \mu_{\mathbf{W}_2}$ , the KL divergence between their covariance matrices can be expressed as:

$$D_{\text{KL}}(\boldsymbol{\Sigma}_{\mathbf{W}_1} || \boldsymbol{\Sigma}_{\mathbf{W}_2}) = \frac{1}{2} \left[ \text{tr} \left( \boldsymbol{\Sigma}_{\mathbf{W}_2}^{-1} \boldsymbol{\Sigma}_{\mathbf{W}_1} \right) - \ln \left( \frac{|\boldsymbol{\Sigma}_{\mathbf{W}_1}|}{|\boldsymbol{\Sigma}_{\mathbf{W}_2}|} \right) \right] \quad (8.3)$$

The symmetrized KL divergence between positive semi-definite matrices (e.g. covariances) also acts as the square of a distance [291] (see the supplementary for further details, including descriptions of other covariance similarity measures). We consider both Equation 8.2 and Equation 8.3 for fusing convolutional layers, self-attention layers and fully-connected layers. The KL is an asymmetric measure, therefore the divergence in both directions can be used to assign a weight to each layer pair in LF.

**OPTIMAL TRANSPORT & WASSERSTEIN DISTANCE** Unlike an all-pair distance such as CA, Wasserstein (WS) distance can also be used to find the optimal cost, also known as the optimal flow between two distributions. Unlike, other distance measures, WD tries to keep the geometry of the distributions intact when interpolating and measuring between distributions. Unlike CA and other baseline measures, WS is invariant to layer permutations and like CA, it also accounts for mutual dependencies between parameters in any arbitrary layer. In this work, we consider the WD between adjacent row-normalized parameter pairs  $\text{softmax}(\mathbf{W}_1, \mathbf{W}_2)$  (i.e. multisets) in a Euclidean metric space. Given two multi-sets  $\mathbf{W}_1, \mathbf{W}_2 \subset \mathbf{W}$ , of size  $d = |\mathbf{W}_1| = |\mathbf{W}_2|$  with corresponding empirical distributions  $P_{\mathbf{W}_1}$  and  $P_{\mathbf{W}_2}$ , the WS distance is defined as Equation 8.4. However, computing WS distance is  $\mathcal{O}(N^3)$  using the standard Hungarian algorithm, which is intractable for large  $\theta$ .

$$D_{W_p}(P_{\mathbf{W}_1}, P_{\mathbf{W}_2}) = \inf_{\pi} \left( \sum_{i=1}^d \|P_{\mathbf{W}_1^i} - P_{\mathbf{W}_2^{\pi(i)}}\|^p \right)^{1/p} \quad (8.4)$$

One way to tradeoff this computational burden is to assume that the weights are i.i.d and normally distributed at the expense of disregarding mutual dependencies learned throughout training. According to Lyapunov’s central limit theorem [228], we can assume the the weights in a layer are normally distributed as the number weights goes to infinity. Hence, if  $P_{\mathbf{W}_1} = N(\mu_{\mathbf{W}_1}, \boldsymbol{\Sigma}_1)$  and  $P_{\mathbf{W}_2} = N(\mu_{\mathbf{W}_2}, \boldsymbol{\Sigma}_2)$  we can express the 2-WS distance as Equation 8.5, also known as the Bures metric<sup>1</sup>.

$$\begin{aligned} D_{W^2}(P_{\mathbf{W}_1}, P_{\mathbf{W}_2}) &= \|\mu_{\mathbf{W}_1} - \mu_{\mathbf{W}_2}\|^2 + \mathbb{B}^2(\boldsymbol{\Sigma}_{\mathbf{W}_1}, \boldsymbol{\Sigma}_{\mathbf{W}_2}), \\ \mathbb{B}^2(\boldsymbol{\Sigma}_{\mathbf{W}_1}, \boldsymbol{\Sigma}_{\mathbf{W}_2}) &= \text{tr}(\boldsymbol{\Sigma}_{\mathbf{W}_1}) + \text{tr}(\boldsymbol{\Sigma}_{\mathbf{W}_2}) - \\ &\quad 2\text{tr} \left[ \left( \sqrt{\boldsymbol{\Sigma}_{\mathbf{W}_1}} (\boldsymbol{\Sigma}_{\mathbf{W}_2} \sqrt{\boldsymbol{\Sigma}_{\mathbf{W}_1}}) \right) \right] \end{aligned} \quad (8.5)$$

Although we focus on the Bures metric in our experiments, an alternative approach is to find a set of cluster centroids in  $\mathbf{W}_1$  and  $\mathbf{W}_2$  as  $C_{\mathbf{W}_2}$  and

<sup>1</sup> Often used in quantum physics for measuring quantum state correlations [110].

$C_{\mathbf{w}_2}$  and compute  $W(P_{C_{\mathbf{w}_1}}, P_{C_{\mathbf{w}_2}})$ . In this approach the centroids are converted to an empirical distribution  $P_{C_\theta}$  such  $c \ll d$  such that a  $\mathcal{O}(N^3)$  cost is feasible for computing during retraining steps. Alternatively, we could avoid softmax normalization and directly compute  $\mathbb{W}(C_{\mathbf{w}_1}, C_{\mathbf{w}_2})$  on both discrete sets. Lastly, we note that when fusing layers with 2-WS distance, the fusion occurs between aligned weights given by the cost matrix. Hence, it is not only used to identify top- $k$  most similar layers, but the cost matrix also aligns which weights in the layer pair are fused.

---

**Algorithm 3** Layer Fusion algorithm. (LFA)
 

---

```

1: input: alignment function  $D$ , parameters  $\theta$ ,  $k$  layers to merge.
2:  $\mathbf{S} = \mathbf{0}_{|\theta|, |\theta|}$ 
3: for  $\mathbf{W}_i$  in  $\{\theta_i\}_{i=1}^{|\theta|}$  do
4:   for  $\mathbf{W}_j$  in  $\{\theta_i\}_{j=i}^{|\theta|}$  do
5:     if param_type( $\theta_i$ ) == param_type( $\theta_j$ ) then
6:        $\mathbf{S}_{ij} = D(\text{vec}(\mathbf{W}_i), \text{vec}(\mathbf{W}_j))$ 
7:     end if
8:   end for
9: end for
10:  $\{I\}$  list of tuples of matrix indices  $\arg \max_k \mathbf{S}$ 
11: for  $(i_1, \dots, i_k)$  in  $\{I\}$  do
12:    $\tilde{\mathbf{W}}_{ij} \sim \mathbb{B}(\mathbf{W}_i, \mathbf{W}_j)$ 
13:    $\tilde{\mathbf{b}}_{ij} \sim \mathbb{B}(\mathbf{b}_i, \mathbf{b}_j)$ 
14:    $\theta_i = \theta_j = \{\tilde{\mathbf{W}}_{ij}, \tilde{\mathbf{b}}_{ij}\}$ 
15: end for
16: return  $\theta$ 

```

---

### 8.3.3 Fusing Layers

After choosing the top- $k$  layer pairs to merge, we then consider 3 ways to fuse the layers: (1) freeze one of the two layers and freeze the gradients for one of the two layers, (2) take the mean between layer pairs and compute backprop on the averaged layer pair and (3) sample and mix between the layers. Merging layer pairs with (1), refers to lines 12-14 in Algorithm 3. Choosing the layers to fuse for (1) is based on which of the two is closest to the middle layer of the network. This is motivated by previous work that showed layers closer to the input and output are generally more salient [482]. When using Jensen-Shannon divergence for choosing top- $k$  layers, we use the divergence asymmetry for choosing which layer is frozen. This is achieved by taking the parameter  $\gamma$  between the Jensen-Shannon divergence of two layers in both directions to control a weighted gradient. We express the backpropagation when using LF with Jensen-Shannon divergence in terms of KL-divergences as shown in Equation 8.6, where  $\tilde{\mathbf{W}}_{ij}$  is a mixture distribution between  $\mathbf{W}_i$  and  $\mathbf{W}_j$  with a weighted gradient  $\partial \mathcal{L} / \partial \tilde{\mathbf{W}}_{ij}$  that represents the gradient for

both  $\mathbf{W}_i$  and  $\mathbf{W}_j$ . Here,  $\gamma = \frac{1}{2} \left( D_{\text{KL}}(\mathbf{W}_i || \bar{\mathbf{W}}_{ij}) + D_{\text{KL}}(\mathbf{W}_j || \bar{\mathbf{W}}_{ij}) \right)$  is a coefficient to balance the weighted average of gradients between the chosen layers.

Thus, for the backward pass of a frozen layer from a given top- $k$  pair, we still compute its gradients which will influence how its original pair will be updated. This constraint ensures that the original pair that were most similar for a given compression step remain relatively close throughout retraining. The layer pair are then averaged at test time to reduce network size, while maintaining similarity using the aforementioned JS divergence gradient constraint.

#### 8.4 EXPERIMENTAL DETAILS

We focus on Transformer-based models for language modeling on the WikiText-2 dataset [277]. For large models in NLP such as BERT [90], OpenAI-GPT, GPT2 [349] and Transformer-XL [463], we freeze or combine layer weights of each multi-attention head component and intermediate dense layers, dramatically reducing the respective number of layer and weights.

$$\frac{\partial \mathcal{L}}{\partial \bar{\mathbf{W}}_{ij}} := \gamma \left( \frac{\partial L}{\partial \mathbf{W}_i} \right) + (1 - \gamma) \left( \frac{\partial L}{\partial \mathbf{W}_j} \right) \quad (8.6)$$

For Equation 8.6, the gradients are averaged for the layer pair that are chosen for layer fusion. The alternative fusion method (i.e mixing) interpolates between hidden representations that are most similar, which can be viewed as a stochastic approach of the aforementioned JS divergence, with the aim of removing redundancy in the network. We denote a pair of randomly mixed layers as  $\tilde{\mathbf{W}}_\ell^i \sim \mathbf{B}(\mathbf{W}_\ell^i, \mathbf{W}_{\ell+1}) \quad \forall i \in n_\ell$ . Note that we only mix between pairs of weight matrices, the bias terms are averaged  $(\mathbf{b}_\ell + \mathbf{b}_{\ell+1})/2$ . We then compute backpropagation on  $\tilde{\theta}_\ell^i$  instead of the original unmixed layer pair  $(\theta_\ell^i, \tilde{\theta}_{\ell+1}^i)$  i.e mixing is carried out before the forward pass. We summarize the standard retraining procedure in Algorithm 4 where LFA performs similarity and merging from Algorithm 3.

---

**Algorithm 4** Retraining algorithm. (RA)

---

```

1: input: batch size  $M$ , number of batches  $N$ , compression epoch
   interval  $N_c$ , pretrained  $f_\theta$ , alignment function  $D$ .
2: for an epoch  $\{\epsilon_i\}_{i=1}^N$  do
3:   for  $(\mathbf{X}, \mathbf{y})$  sampled minibatch  $\{\mathbf{B}_i\}_{i=1}^N$  do
4:      $\hat{\mathbf{y}} = f_\theta(\mathbf{X})$ 
5:     Update gradients on  $f_\theta$  to minimize  $\ell_{\text{CE}}(\hat{\mathbf{y}}, \mathbf{y})$ 
6:   end for
7:   if  $\epsilon_i \bmod N_c$  then
8:      $\theta = \text{LFA}(\theta, D, k)$ 
9:   end if
10: end for
11: return  $f_\Theta(\cdot)$ 

```

---

For image classification on CIFAR10, we report results for ResNet, ResNet-ELU [383], Wide-ResNet [479] and DenseNet [174]. We are particularly interested in ResNet architectures, or more generally, ones that also use skip connections. This is motivated by Veit, Wilber, and Belongie [426] which found that deleting or permuting residual blocks can be carried out without much degradation in performance in a pretrained ResNet.

#### 8.4.1 Compression Without Retraining

For magnitude-based pruning, we prune a percentage of the weights with the lowest magnitude. This is done in one of two ways: a percentage of weights pruned by layer (layer pruning), or a percentage of the network as whole (global pruning). For quantization, we use k-means whereby the number of clusters for a given layer is specified as a percentage of the original size of that layer (i.e number of parameters in the tensor). For tensor decomposition, we reduce the number of parameters by approximating layers with a lower rank using SVD. Specifically, we use randomized truncated SVD [139] where QR factorization on  $\mathbf{W}_\ell$  such that  $\mathbf{Q}_\ell^T \mathbf{W}_\ell = \mathbf{R}_\ell$  where  $\mathbf{Q}_\ell$  are the orthonormal columns of  $\mathbf{W}_\ell$ . Randomized methods are used to approximate the range of  $\theta_\ell$  and reduce computation from  $\mathcal{O}(\min(n_{\ell-1}n_\ell^2, n_{\ell-1}^2n_\ell))$  to  $\mathcal{O}(n_{\ell-1}n_\ell \log(k))$  where  $k$  represents the approximate rank of  $\theta_\ell$ . We also perform dimensionality reduction on the layers by using 1-hidden layer denoising autoencoders which use the same activation functions for reconstruction as the original architecture and a mean squared error loss is minimized. The encoder layer of each denoising AE (DAE) is the used in replacement of the original layer. For both truncated SVD and DAE, this is carried out sequentially from bottom to top layer so that the reconstruction of a given layer  $l$  also accounts for cascade approximation errors of dimensionality reduction from previous layers. We refer to this type of layer reconstruction technique as *student rollout*

because the pretrained teacher network is iteratively rolled out and reconstructed from the first layer to the last.

### 8.4.2 Layer Fusion & Compression ReTraining

For retraining we consider two main schemes: (1) for each retraining step we carry out network compression (e.g via pruning), retrain the resulting network and iteratively repeat until the final epoch, and (2) in the case where network compression leads to non-zero weights (e.g LF), we freeze the network weights apart from those which have been identified for LF in which case we retrain before tying.

Layer averaging, mixing and freezing are experimented with for fusing layers. To maintain uniformity across each compression step, we prune, quantize, fuse and decompose a percentage of the weights as opposed to using other criteria such as the weight magnitude thresholding.

This ensures comparability between compression methods e.g thresholding weights in pruning does not have a direct equivalent to quantization or weight decomposition, unless network reduction is proportional to the number of weights pruned via thresholding.

## 8.5 RESULTS

This section describes are results of layer fusion applied to CNNs and Transformers on image classification and language modeling respectively.

### 8.5.1 Image Classification

Figure 8.1(a) shows the results of pruning, quantization, weight decomposition and our proposed LF *without* any retraining. A general observation is that an exponential decline in performance occurs at

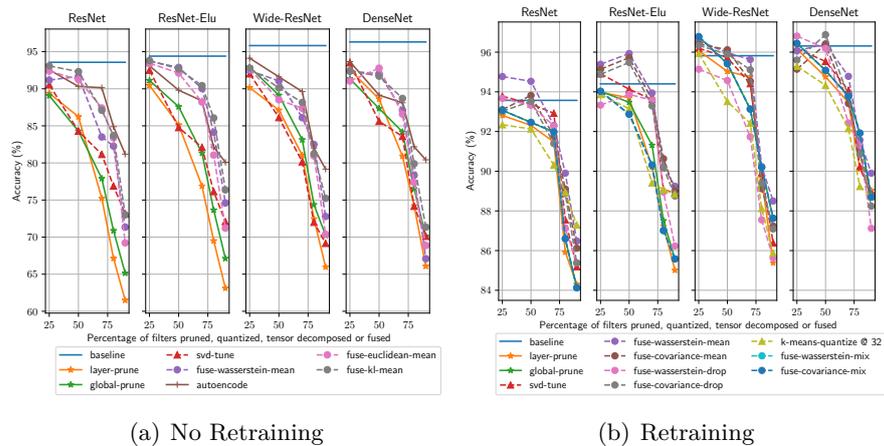


Figure 8.1: CIFAR-10 Test Accuracy With And Without Retraining

Table 8.1: CIFAR-10 Test Accuracy with WS-Based CNN Layer Fusion

		Res		Res-ELU		Wide-Res		DenseNet	
<b>Orig.</b>		93.75	-	94.40	-	95.82	-	96.31	-
<b>Mean</b>	25%	92.39	94.77	93.45	95.39	92.66	96.57	91.04	96.06
	50%	91.24	94.53	92.12	95.93	88.51	95.97	92.78	96.42
	75%	87.41	92.30	88.20	93.94	87.36	95.63	86.58	94.78
	80%	83.40	89.90	81.06	90.23	80.94	90.23	77.36	88.50
	90%	<b>69.22</b>	<b>86.48</b>	<b>71.20</b>	<b>89.24</b>	<b>70.38</b>	<b>89.90</b>	<b>68.87</b>	<b>91.57</b>
<b>Freeze</b>	25%	91.17	93.67	93.71	93.33	92.45	95.14	92.36	96.81
	50%	91.67	93.32	92.88	93.87	91.06	94.57	92.03	96.19
	75%	83.50	92.28	90.02	93.58	86.10	91.73	87.11	92.43
	80%	82.27	87.12	84.12	88.95	82.49	87.55	78.35	85.63
	90%	<b>71.34</b>	<b>85.38</b>	<b>74.60</b>	<b>86.23</b>	<b>72.78</b>	<b>85.63</b>	<b>67.09</b>	<b>87.12</b>
<b>Mixing</b>	25%	93.67	93.22	93.33	94.03	95.14	96.78	96.81	96.44
	50%	93.32	92.46	93.24	93.87	94.57	95.42	96.19	95.08
	70%	92.28	91.98	90.31	93.58	91.73	93.13	92.43	93.78
	80%	84.12	90.60	85.58	88.95	87.55	90.20	91.32	91.92
	90%	<b>73.2</b>	<b>86.13</b>	<b>73.50</b>	<b>85.58</b>	<b>80.63</b>	<b>87.63</b>	<b>78.12</b>	<b>88.70</b>

around 70% (some variance depending on the compression method) of the original network is compressed. For example, fusing layers using the WS distance for alignment allows accuracy to be closer to the original networks accuracy up to 70%. In contrast, pruning convolutional layers in ResNet models leads to a faster accuracy drop. This is surprising given that unstructured pruning is less restrictive, when applying LF to CNN architectures. We also allow filters from the same layer to be fused, in comparison to dense layers in self-attention blocks for Transformers.

Figure 8.1(b) shows results of the compression techniques *with* retraining. We see the results of model compression methods retraining on CIFAR-10 for ResNet-50, ResNet-50 with exponential linear units (ELUs), Wide-ResNet and DenseNet. We test each combination of layer pairs for averaging layers as  $\tilde{\theta}_i = \tilde{\theta}_j = (\theta_i + \theta_j/2)$  where  $\binom{L}{2}$  are the total number of layers (e.g 24 layers results in 276 possible pairs). The performance change is measured from the original network when layer averaging by choosing the top-k ( $k \ll L$ ) layers and measuring which averaged layer pair produced the smallest difference in accuracy when compared to the original network. When the same layer within the top-k chosen layers is coupled with more than one other layer, we take the mean of multiple pairs to reduce computation to  $2\binom{L}{k}$ . We find a reoccurring pattern that early on retraining, we observe up to a reduc-

tion of 25% of the network improves the results, and even up to 25% - 50% in some cases (e.g global pruning and layer pruning). From 75% we see a significant decrease in performance, typically 2-4% drop in accuracy percentage points across each model. Given an allowance of  $N$  retraining epochs, allocating the amount of model compression for each compression step is a critical hyperparameter. Concretely, less retraining time is necessary for during initial model compression, whereas past a compression ratio of 3.33 (i.e 75%), the interval between retraining steps should become larger. This is highlighted in bold across Table 8.1, where left subcolumns are with no retraining and right subcolumns are with retraining. Here, the top two cells show iterative pruning results, “Randomized SVD” applies tensor decomposition with retraining, “Denoising Autoencoder” reconstructs the output of layers to a lower dimension, “Layer Averaging” and “Layer Freezing” are two variants of layer fusion and finally “Global WS-LF” and “Adjacent WS-LF” apply layer fusion using the Wasserstein distance metric where “Global” consider all layer pairs while “Adjacent” only considers tying adjacent layer pairs.

For all fusion types (mean, freezing and mixing), we find a significant increase in accuracy after retraining. Mean LF using the WS-2 distance outperforms freezing layers, while random layer mixing performs comparably to averaging. Layer mixing interpolates between neurons a top- $k$  most similar layer pair. Hence, it can change the sign of some of the original incoming weights into the resulting mixed layer. Therefore, it is somewhat surprising that accuracy has remained relatively high, suggesting that similar layers have weights with a shared sign, not only a similar magnitude.

### 8.5.2 Language Modeling

We begin by showing how all compression methods suffer in performance when no retraining is used in Table 8.2. At high compression rates, all compression methods, including out LF techniques (i.e Layer Averaging, Freezing, Gloval WS-LF and Adjacent WS-LF), cannot recover from performance degradation after a one-shot compression step.

In Figure 8.2, we find an exponential trend in perplexity (note the log-scale y-axis) increase with respect to the compression ratio for layer pruning and global pruning. Interestingly, Transformer-XL can maintain similar performance up to 50% pruning from the pretrained model without any retraining. In contrast, we see that the original OpenAI-GPT is more sensitive and begins to show an exponential increase at 30%. This insight is important for choosing the intervals between compress steps during iterative pruning, likewise for LF and tensor decomposition. Concretely, we would expect that the larger the increase in perplexity between compression steps, the more retraining epochs are needed. Instead of choosing a uniform amount of compression

Table 8.2: WikiText-2 Test Perplexity Without Fine-Tuning Or Retraining.

	Trans-XL	GPT-2	GPT	Trans-XL	GPT-2	GPT
<b>Original</b>	21.28	26.61	67.23	21.28	26.61	67.23
	<b>Layer Pruning via Weight Magnitude</b>			<b>Global Pruning via Weight Magnitude</b>		
@ 10%	21.25	25.44	69.33	21.15	25.04	69.54
@ 20%	21.26	27.02	88.19	21.08	27.03	79.33
@ 30%	22.05	35.87	1452.96	21.54	46.15	140.22
@ 50%	57.12	1627.22	3260.52	53.90	3271.52	2159.42
@ 70%	3147.31	24946.66	21605.02	901.534	13464.17	18068.86
	<b>Randomized SVD</b>			<b>Denosing AutoEncoder</b>		
@ 10%	20.29	25.44	69.33	19.69	23.14	65.14
@ 20%	20.69	27.02	88.19	19.43	24.46	81.08
@ 30%	21.68	35.87	1452.96	20.57	29.07	921.06
@ 50%	64.12	1627.22	3145.41	55.07	1258.05	2654.88
@ 70%	3679.13	26149.57	22140.12	2958.41	19206.78	19035.38
	<b>Layer Averaging (Euclidean Distance)</b>			<b>Layer Freezing (Euclidean Distance)</b>		
@ 10%	21.74	25.78	81.14	23.09	28.70	83.44
@ 20%	22.21	29.74	94.80	25.19	30.88	94.32
@ 30%	25.27	38.90	1903.14	27.81	40.01	97.11
@ 50%	62.04	1807.31	3724.47	64.38	1944.51	3790.12
@ 70%	3695.01	2631.52	29117.82	3583.16	23583.10	30258.78
	<b>Global WS-LF</b>			<b>Adjacent WS-LF</b>		
@ 10%	22.15	25.79	69.29	22.52	25.58	69.90
@ 20%	22.37	27.38	90.70	22.61	27.35	89.77
@ 30%	24.79	38.18	1533.24	22.82	36.11	1493.37
@ 50%	61.68	1690.31	3123.39	59.70	1691.23	3357.02
@ 70%	3201.97	25130.30	22448.15	3198.16	25270.21	21732.58

at each compression step  $N_c$ , more compression are allocated earlier but longer retraining step intervals later.

Figure A.9 shows the similarity between pretrained layers on Transformer-XL using the sum of pairwise Euclidean distances. In general, we can see that closer layers have a smaller Euclidean distance. This more pronounced in the output attention (3) and fully connected layers (4) and slightly more sporadic among query-key-value attention weights (1).

Figure 8.4 shows subfigures of retraining GPT (8.4(a)), GPT-2 (8.4(b)) and Transformer-XL (8.4(c)) with all aforementioned compression methods for GPT, GPT2 and Transformer-XL respectively. Firstly, we find retraining with a sufficient number of compression steps to be worthwhile for drastically reducing the network size while maintaining performance for both structured and unstructured approaches. Past 30% of network reduction we find a weakly linear increase, in contrast to the exponential increase with no retraining. We find that global pruning generally outperforms layer pruning as it doesn't restrict the percentage of weights pruned to be uniform across layers. This suggests that many layers are heavily pruned while others are preserved. This also coincides with findings from [482] that some layers are critical to maintain performance

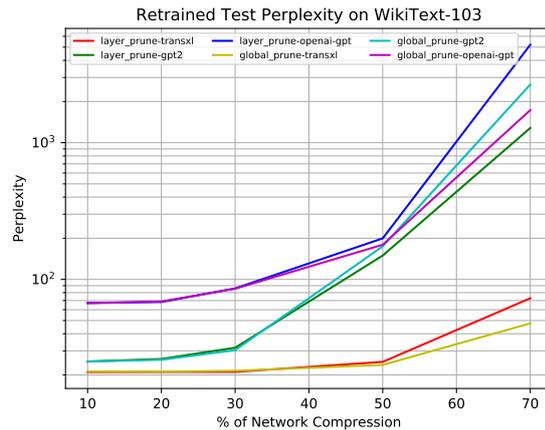


Figure 8.2: Wikitext-2: Pruning Without Retraining

while removing the remaining layers has little effect on generalization. Table 8.3 shows the results of LF for compression ratio of 2 using layer averaging (Mean), layer freezing (Freeze) and mixing layers (-Mix) when ranking weight similarity using Euclidean distance (ED), KL and WS distance and CA. For all models CA produces the best results, slightly outperforming WS. Lastly, we compare the best results of Table 8.3 with *LayerDrop* [104, p. LD], a method that has shown SoTA for structured compression that require no additional overhead, akin to LF. Table 8.4 shows that the best results obtained from LF outperforms LD when evaluating the models at 50% reduction of their original size. For LD, this corresponds to a dropout rate of 0.5 (best results at 0.5 in Fan, Grave, and Joulin [104]) during retraining on WikiText-2 followed by pruning 50% of layers post-retraining. For all 3 architectures, we find our best LF results improve over LD.

**ADDITIONAL OBSERVATIONS** In language modeling, the effects of model reduction typically follow an exponential increase in perplexity for a compression ratio greater than 2 (corresponding to @50%) when no retraining steps are used. Unlike CIFAR10 image classification, language modeling is a structured prediction task that has a relatively large output dimensionality which we posit has an important role in the amount of compression that can be achieved. Yang et al. [462] have noted the *softmax bottleneck* whereby the restriction on the size of the decoder results in information loss when calibrating the conditional distribution, while [28] have also noted the double descent phenomena is dependent on the number of classes. We conjecture that pruning and other such methods can exacerbate this bottlenecking and therefore the compression ratio will be generally lower compared to classification problems with relatively less classes, such as CIFAR-10.



Table 8.4: WikiText-2: Perplexity With *LayerDrop* & *Layer Fusion*

	<b>TransXL</b>	<b>GPT2</b>	<b>GPT</b>
<b>Layer Fusion</b>	11.13	13.71	13.58
<b>LayerDrop</b>	14.30	18.93	21.01

during retraining. Lastly, a clear inflection point was observed in both tasks where performance rapidly decreases for all compression methods and models.

Table 8.3: WikiText-2: Perplexity of LF-Retraining @50% reduction

	Mean	Freeze	Mix
<b>TransXL-KL</b>	12.23	15.02	13.75
<b>TransXL-ED</b>	13.08	17.13	14.88
<b>TransXL-WS</b>	11.48	14.40	12.17
<b>TransXL-CA</b>	<b>11.13</b>	13.97	14.73
<b>GPT2-KL</b>	15.56	19.04	15.87
<b>GPT2-ED</b>	16.03	21.14	16.73
<b>GPT2-WS</b>	<b>13.71</b>	18.31	13.58
<b>GPT2-CA</b>	14.03	21.28	14.59
<b>GPT-KL</b>	23.57	28.01	24.82
<b>GPT-ED</b>	25.07	29.68	24.73
<b>GPT-WS</b>	19.10	23.17	18.90
<b>GPT-CA</b>	<b>18.48</b>	22.01	20.39

---

## ITERATIVE PRUNING IN THE ZERO-SHOT SETTING

---

Pruning aims to reduce model complexity while maintaining performance close to the original unpruned model. While various avenues of research have been explored for iterative pruning, little is known what effect pruning has on zero-shot test performance and its potential implications on the choice pruning criteria. This pruning setup is particularly important for cross-lingual models that implicitly learn alignment between language representations during pretraining, which if distorted via pruning, can lead to poor performance when evaluated on languages it has not observed during fine-tuning on a downstream task. This work highlights that there is a clear performance discrepancy in magnitude-based pruning when comparing standard supervised learning to the zero-shot setting. From this finding, two weight regularizers are proposed that aim to maximize the alignment between units of pruned and unpruned networks to mitigate alignment distortion in pruned cross-lingual models for the zero-shot setting. Namely, (i) a unit-wise weight regularizer that maximizes the cosine similarity of weights between pruned and unpruned networks and (ii) a layer-wise weight regularizer that minimizes the Frobenius norm between the weight tensors of pruned and unpruned networks. Experimental results are provided on cross-lingual tasks for the zero-shot setting using XLM-RoBERTa<sub>Base</sub> where it also found that pruning has varying degrees of representational degradation depending on the language corresponding to the zero-shot test set. This is also the first study that focuses on cross-lingual language model compression.

### 9.1 INTRODUCTION

Deep neural networks (DNNs) have grown increasingly large in the recent years. This has led to models requiring more storage requirements, more resources for training and inference (e.g GPUs and TPUs), longer compute times and larger carbon footprints. This is largely due to the rise of masked Self-Supervised Learning (SSL) which trains DNNs (e.g Transformers in NLP) on a large collection of samples that do not have task labels but instead use a subset of the inputs as labels. Given the aforementioned challenges, it has become more difficult for machine

learning practitioners to use these SSL pretrained models for fine-tuning on downstream tasks. While training tricks such as effective batch sizes, gradient accumulation and dynamic learning rate schedules [169] have improved the efficiency of fine-tuning DNNs under resource constraints, it can still come at a cost e.g gradient accumulation leads to less gradient updates.

Pruning [223, 355] is a type of model compression method [49] that aims to address these shortcomings by zeroing out a subset of weights in the DNN, while maintaining performance close to the original model. Retraining is often carried out directly after each pruning step to recover from pruning induced performance drops, which is referred to as *iterative pruning*. Although, iterative pruning has been extensively studied in the SSL setting [93, 143, 147] and the transfer learning setting [129, 295, 374], little is known about pruning DNNs in the zero-shot setting<sup>1</sup> where a model is required to make predictions on a set of samples from classes that are unobserved during training. One salient example is pretrained cross-lingual language models (XLMs) [68, 216] whereby the model is trained with a masked/translation language model (MLM/TLM) objective to predict tokens for a large set of different languages whereby the objective forces the XLM model to learn similar representations for different languages. After cross-lingual pretraining, the model is further fine-tuned to a downstream task in one language (e.g English) and then evaluated in different languages (e.g Spanish, French, Chinese etc.). In this context, applying current pruning methods can damage the XLM cross-lingual alignment that has been learned during pretraining. Ideally, the aim is to prune XLMs in such a way that avoids this alignment distortion which effects the zero-shot performance of pruned XLMs. Additionally, overfitting to the language used for fine-tuning becomes more of an issue due to the progressive reduction in parameters throughout iterative pruning as the remaining weights are relatively large, moving away from an “aligned” XLM state.

This is an important problem to address as the application of large pretrained models in both natural language and computer vision in the zero shot-setting is of practical importance e.g using XLMs in production for multiple languages by only requiring annotations in a single language for fine-tuning, making predictions on unseen classes at test time from pretrained visual representations [48] using only semantic description (i.e label similarity to known classes) or zero-shot predictions in pretrained multi-models such as CLIP [347].

Hence, this work addresses the *alignment distortion* pruning problem by introducing AlignReg, a class of weight regularizers that force pruned models to have parameters that point in a similar direction to the parameters of the original pretrained network. To our knowledge, this is

<sup>1</sup> Here, zero or one-shot is the conventional usage of the meaning (i.e number of samples per class), not one-shot pruning [227] which is the number of pruning steps used during retraining.

the first study on how iteratively pruned models perform in the zero-shot setting and how the solution differs from solutions found in the non-zero shot setting. Our findings have a strong practical implication as well-established pruning criteria may not be suitable given the discrepancy between zero-shot performance and the typically reported non-zero shot performance. Moreover, our proposed weight regularizer improves overall pruning generalization in zero-shot cross-lingual transfer. Below, a summary of our **contributions** are listed.

- The first analysis of pruning cross-lingual models, how this effects zero-shot cross-lingual transfer and performance differences to pruning in the SSL setup.
- A weight regularizer that mitigates alignment distortion by minimizing the layer-wise Frobenius norm or unit similarity between the pruned model and unpruned model, avoiding overfitting to single language task fine-tuning.
- A post-analysis of weight distributions after pruning and how they differ across module types in Transformers.

## 9.2 RELATED WORK

Below briefly describes relevant work on regularization-based pruning methods, other non-magnitude based pruning methods and how masked language modeling (MLM) implicitly learns to align cross-lingual representations.

**Regularization-based pruning.** Pruning can be achieved by using a weight regularizer that encourages network sparsity. Three well-established regularizers are  $L_0$  [264],  $L_1$  regularization [257, 465] and the commonly used  $L_2$  regularization for weight sparsity [143, 145]. Wang et al. have proposed an  $L_2$  regularizer that increases in influence throughout retraining and shows the increasing regularization improves pruning performance. For structured pruning where whole blocks of weights are removed, Group-wise Brain Damage [224] and SSL [440] propose to use Group LASSO [477] to learn structured solutions.

**Importance-based pruning.** Magnitude-based pruning (MBP) relies on the assumption that weight or gradient magnitudes have correlation with its importance to the overall output of the network. Mozer and Smolensky instead use a learnable gating mechanism that approximates layer importance, finding that weight magnitudes reflect importance statistics. To measure weight importance as the difference in loss between pruned and unpruned network, LeCun, Denker, and Solla approximate this difference with a Taylor series up to the second order. This involves the product of the gradient and weight magnitude in the 1st term and an approximation of the Hessian and the square of the weight magnitude for the second term. They find this improves iterative

pruning performance as convergence is faster between pruning steps, leading to higher compression rates. However, computing the Hessian and even its approximations [96, 147, 223, 393, 433] can significantly slow down retraining. In this work, the aim is to avoid the requirement of computing the Hessian or approximations thereof, as it is not scalable for models such as XLM-R [68]. Park et al. have extended MBP to block approximations to avoid pruning lowest weight magnitudes that may be connected to weights in adjacent layers that have high weight magnitude. Lee et al. have provided a method to automatically choose the sparsity of layers by using the rescaled version of weight magnitude to incorporate the model-level distortion incurred by pruning.

IMPLICITY CROSS-LINGUAL ALIGNMENT IN PRETRAINED MLMS  
 Prior work [450] has found that multilingual MLM (i.e training with an MLM objective with concatenated text for multiple languages) naturally leads to models with strong cross-lingual transfer capabilities. Additionally, they find that this is also found for monolingual models that do not share vocabulary across monolingual corpora and the only requirement is that weight sharing is used in the top layers of the multi-lingual encoder. In the context of our work, it is desired to bias our fine-tuned and iteratively pruned model to have similar geometric properties and symmetries to these pretrained MLMs to preserve zero-shot cross-lingual transfer generalization.

### 9.3 METHODOLOGY

This section describes how the proposed AlignReg weight regularizers can improve pruning performance in both supervised learning and zero-shot pruning settings. The two AlignReg regularizers, namely, a neuron correlation-based regularizer (cosine-MBP) and Frobenius layer-norm regularizer (frobenius-MBP) are focused on. Below, begins with an introduction of the proceeding notation. Let  $\mathcal{D} := \{X_i, y_i\}_{i=1}^D$  where each  $X_i$  of  $D$  training samples consists of a sequence of vectors  $X_i := (\mathbf{x}_1, \dots, \mathbf{x}_n)$  and  $\mathbf{x}_i \in \mathbb{R}^d$  (e.g  $d = 512$ ). For structured prediction (e.g NER, POS),  $y_i \in \mathbb{R}^{n \times c}$  and for single and pairwise sentence classification,  $y_i \in \mathbb{R}^c$  where  $c$  is the number of classes. Let  $\theta = (\theta_1, \dots, \theta_L)$  be the parameters of a pretrained network  $f$  with  $L$  layers, where  $\theta_l$  refers to the parameters, including weight matrix  $\mathbf{W}_l$  and bias  $b_l$ , at layer  $l$ . Let  $f_{\tilde{\theta}}$  be a network with parameters  $\tilde{\cdot}$  consisting of weights  $\tilde{\mathbf{W}}_l \in \mathbb{R}^{N_{l-1} \times N_l}$  and bias  $\tilde{b}_l \in \mathbb{R}^{N_l}$  where  $N_l$  is the number of units in the  $l$ -th layer. Here,  $\tilde{\mathbf{W}}_l := \mathbf{W}_l \mathbf{M}_l$  where  $\mathbf{M}$  is the pruned mask. For MBP [191] the weights of  $\mathbf{W}_l$  are removed,  $\forall l \in L$  with the smallest absolute weight magnitude until a specified percentage  $p$  of weights are removed. Note that this is a layer-wise process and requires the pruned weights to be masked with  $\mathbf{M}_l$  which has 0 entries corresponding to weights to be pruned and 1 entries for unpruned weights  $\mathbf{W}_l$ . Global MBP can also be

used whereby the weights  $\{\mathbf{W}_l\}_{l=1}^L$  are first vectorized and concatenated prior to choosing  $p$  lowest weight magnitudes. The main difference of global MBP when compared to layer-wise MBP is that percentage of weights removed in each layer can vary.

The loss during iterative pruning can then expressed as,

$$\mathcal{L}_\theta(\mathbf{X}, \mathbf{y}) := \mathbb{E}_z \left[ \frac{1}{D} \sum_{i=1}^D \ell_{ce}(f^z(\mathbf{X}_i), \mathbf{y}_i) + \lambda \|\tilde{\cdot}\|_0 \right], \quad (9.1)$$

where  $\lambda$  controls the influence of the weight magnitude regularization. Below describes our proposed AlignReg.

### 9.3.1 AlignReg *Between Pruned and Unpruned Layers*

AlignReg aligns weights unit-wise or layer-wise between unpruned and pruned networks. By constraining the angle of the parameter vector of the pruned network to point in a similar direction to the unpruned network, one aims to preserve the inherent cross-lingual alignment while iteratively pruning. To apply *AlignReg* to linear layers within Transformers, the pairwise cosine similarity between pairs of pruned weights  $\tilde{\mathbf{W}}_l \subset \tilde{f}$  and unpruned weights  $\mathbf{W} \subset f$  for all  $l$ -th layers are computed. For  $\mathbf{W}_l \in \mathbb{R}^{N_{l-1} \times N_l}$  of the  $l$ -th layer, the average weight correlation is defined as

$$\rho(\tilde{\mathbf{W}}_l, \mathbf{W}_l) = \frac{1}{N_l} \sum_{i=1}^{N_l} \frac{|\mathbf{W}_{li}^\top \tilde{\mathbf{W}}_{li}|}{\|\mathbf{W}_{li}\|_2 \|\tilde{\mathbf{W}}_{li}\|_2} \quad (9.2)$$

where  $\mathbf{W}_{li}$  is  $i$ -th column of the matrix corresponding to the  $i$ -th unit of the  $l$ -th layer. Intuitively,  $\rho(\mathbf{W}_l, \tilde{\mathbf{W}}_l)$  is the average cosine similarity between weight vectors of the same unit at the  $l$ -th layer of the pruned and unpruned network. Adding AlignReg to the objective results in Equation (9.3),

$$\mathcal{L}_\theta(\mathbf{X}, \mathbf{y}) := \ell_{ce}(f_{\tilde{\theta}}(\mathbf{X}), \mathbf{y}) + \frac{\lambda}{L} \sum_l \rho(\tilde{\mathbf{W}}_l, \mathbf{W}_l) \quad (9.3)$$

where  $\lambda \in [0, \infty)$  controls the importance of AlignReg relative to the main cross-entropy loss  $\ell_{ce}(\cdot, \cdot)$ . The gradient of the loss w.r.t to  $\theta$  is then expressed as equation (9.4),

$$\nabla_\theta \mathcal{L}_\theta(\mathbf{X}, \mathbf{y}) = \nabla_{\tilde{\theta}} \ell_{ce}(f_{\tilde{\theta}}(\mathbf{X}), \mathbf{y}) + \frac{\lambda}{L} \sum_l \frac{\partial \rho(\tilde{\mathbf{W}}_l, \mathbf{W}_l)}{\partial \tilde{\mathbf{W}}_l} \quad (9.4)$$

where  $\frac{\partial \rho(\tilde{\mathbf{W}}_l, \mathbf{W}_l)}{\partial \tilde{\mathbf{W}}_l}$  is a function of the ‘2-norm of the matrices in  $\mathbf{W}_l$ . For the element  $\mathbf{W}_{l,(i,j)}$  of  $i$ -th row and  $j$ -th column in  $\mathbf{W}_l$ , we have

$$\frac{\partial \rho(\tilde{\mathbf{W}}_l, \mathbf{W}_l)}{\partial \tilde{\mathbf{W}}_{l,(i,j)}} = \frac{1}{N_l - 1} \sum_{j=1}^{N_l} \left( \text{sign}(\mathbf{W}_{l,(j)}^\top \tilde{\mathbf{W}}_{l,(j)}) \left[ \frac{\tilde{\mathbf{W}}_{l,(i,j)}}{\|\mathbf{W}_{l,(j)}\|_2 \|\tilde{\mathbf{W}}_{l,(j)}\|_2} - \frac{\mathbf{W}_{l,(i,j)} \mathbf{W}_{l,(j)}^\top \tilde{\mathbf{W}}_{l,(j)}}{\|\mathbf{W}_{l,(j)}\|_2^3 \|\tilde{\mathbf{W}}_{l,(j)}\|_2} \right] \right) \quad (9.5)$$

where  $\mathbf{W}_{l,(j)}$  and  $\tilde{\mathbf{W}}_{l,(j)}$  are  $j$ -th column in  $\mathbf{W}_l$  and  $\tilde{\mathbf{W}}_l$ , respectively. Thus, this regularization favors solutions with high cosine similarity between units of pruned and unpruned networks. One also considers a layer-wise  $\rho(\mathbf{W}, \tilde{\mathbf{W}})$  that relaxes the unit-level alignment to whole layers. This is partially motivated due to the fact neural networks can exhibit similar output activation behavior even when neuron weights have been permuted within the layer [44]. To perform this, Equation (9.2) is applied with vectorized weights  $\rho(\text{vec}(\tilde{\mathbf{W}}_l), \text{vec}(\mathbf{W}_l))$  and the subsequent partial derivatives in Equations (9.4) and (9.5) are applied for updating  $\tilde{\mathbf{W}}_l$ .

RELAXING UNIT-WISE ALIGNREG TO A LAYER-WISE FROBENIUS DISTORTION FORMULATION Thus far, the application of cosine similarity as a measure of similarity between unpruned and pruned weights of the same units has been described. However, this may be a strict constraint, particularly at high compression rates where the remaining weights for a unit are forced to have higher norms to allow zeroed weights. Hence, an alternative measure is the layer-wise Frobenius norm (Frobenius-MBP) of the difference between weights  $\|\mathbf{W} - \tilde{\mathbf{W}}\|_F$ . MBP itself can be viewed in terms of minimizing the Frobenius distortion [96, 143] as  $\min_{\mathbf{M}: \|\mathbf{M}\|_0=p} \|\mathbf{W} - \mathbf{M} \odot \mathbf{W}\|_F$  where  $\odot$  is the Hadamard product,  $\|\cdot\|_0$  denotes the entrywise 0-norm, and  $p$  is a constraint of the number of weights to remove as a percentage of the total number of weights for that layer.

In the zero-shot setting, the out-of-distribution Frobenius distortions need to be accounted for, such as *alignment distortion* in XLMs occurring due to pruning and overfitting to a single language. Taking the view of Frobenius distortion minimization when using our weight regularizer, this can be reformulated to include frobenius-MBP as,

$$\min_{\mathbf{M}: \|\mathbf{M}\|_0=p} \left[ \|\mathbf{W} - \mathbf{M} \odot \mathbf{W}\|_F + \lambda \|\mathbf{W}^T - \mathbf{M} \odot \mathbf{W}\|_F \right] \quad (9.6)$$

where  $\mathbf{W}^T$  are the weights from the pretrained model prior to fine-tuning that is cross-lingually aligned from the Masked Language Modeling (MLM) pretraining objective. In our experiments,  $\lambda = 5 \times 10^{-4}$  which is identical to the setting when using  $L_2$  regularization, which has the same range as the Frobenius norm.

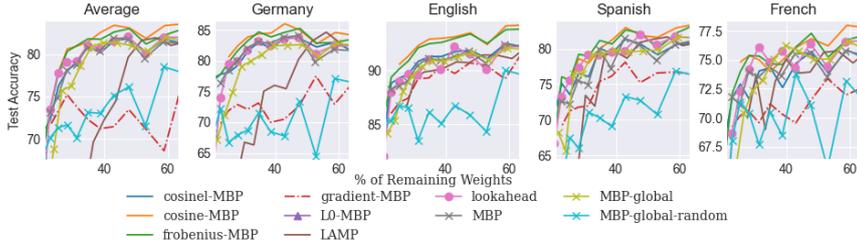


Figure 9.1: English and Zero-Shot Test Accuracy on News Classification.

### 9.3.2 Connections to Knowledge Distillation

Knowledge distillation works by using outputs of the last layer [162] or intermediate layers [360] as additional soft targets. AlignReg regularizers instead operate directly on minimizing a divergence or distance between weight tensors as opposed to their corresponding output activations. Hence, AlignReg does not necessarily need training data as it operates directly on aligning weight tensors. Since the networks that are used for alignment are architecturally identical, it can be shown that maximizing weight similarity is equivalent to minimizing distance between their corresponding output activations [360] when the norm of input  $Z$  is smaller than the output range of  $\sigma$ . For our experiments, XLM-RoBERTa<sub>Base</sub> is used, which contains Gaussian Linear Error Unit (GeLU) activation functions, formulated as  $\sigma(\mathbf{Z}_{li}) := \mathbf{Z}_{li}/2(1.0 + \text{erf}(\mathbf{Z}_{li}/\sqrt{2.0}))$  where  $\text{erf}$  is an error function,  $\sigma(\cdot)$  is a monotonic activation function and  $\mathbf{Z}_{li}$  is the input vector. The GeLU activation has the properties that for  $z > 0$  it is equivalent to the ReLU activation and  $\mathbf{Z}_{li} \leq 0$  it tends to -1. For  $\mathbf{Z}_{li} > 0$ ,  $\|\mathbf{Z}_{li}\|_2 \leq 1$  and a monotonic piecewise linear function  $\sigma(\cdot)$ , the following inequality holds.

$$\|\mathbf{W}_{li} - \mathbf{M}_{li} \odot \mathbf{W}_{li}\|_F \leq \|\sigma(\mathbf{Z}_l \mathbf{W}_{li}) - \sigma(\mathbf{Z}_l \mathbf{M}_{li} \odot \mathbf{W}_{li})\|_F \quad (9.7)$$

Hence, minimizing the Frobenius distortion of pruned and unpruned weights is equivalent to minimizing the Mean Squared Error (MSE) between output activations, as is the knowledge distillation method used for FitNets [360]. In contrast, KD using FitNets encourages the student network to have activation outputs that are the same as the teacher with permutation invariance on the units incoming weights, not restricting the weights to be similar.

## 9.4 EXPERIMENTAL SETUP

**DATASETS.** Experiments are performed on multilingual tasks from the XGLUE benchmark [238] with pretrained XLMR<sub>Base</sub>. This covers pairwise classification (QAM, QADSM, WPR, XNLI), sentence classification (NC) and structured prediction (NER and POS) tasks.

ITERATIVE PRUNING DETAILS. XLM-R<sub>Base</sub> is tokenized the input with SentencePiece BPE tokenizer [381] with a vocabulary of 250K tokens. For structured prediction tasks (POS and NER), a single layer feed-forward (SLFF) token-level classifier is used on top of XLMR and for sentence-level task a SLFF sentence-level classifier is used. The batch size is 32, the learning rate is  $5 \cdot 10^{-6}$  and the maximum sequence length is set to 256 for all tasks, except for POS in which a learning rate of  $2 \cdot 10^{-5}$  and max sequence length of 128 are used. A pruning step after each 15 training epochs is carried out, uniformly pruning 10% of the parameters at each pruning step. This work omits the pruning of embeddings, layer normalization parameters and the classification as they account for a relatively small number of the total parameter count (<1%) and play an important role in XLM generalization. Although prior work have suggested non-uniform pruning schedules (e.g cubic schedule [494]), no major differences to uniform pruning were observed in preliminary experiments. Each task is trained with English data only and evaluated on all available languages for that task. Hence, it is expected that the percentage of achievable compression to be lower as performance in the zero-shot cross-lingual setting to be more difficult than the monolingual setting (e.g GLUE tasks).

PRUNING BASELINES. Below lists our pruning baselines.

- **Random Global Pruning** (MBP-global-random) - pools all layer weights together and randomly chooses weights to remove.
- **Layer-wise Magnitude Pruning** (MBP) - for each layer, prunes the weights with the lowest absolute value (LAV).
- **Global Magnitude Pruning** (MBP-Global) - prunes weights with LAV anywhere in the network.
- **Layer-wise Gradient Magnitude Pruning** (Gradient-MBP) - for each layer, prunes the weights with LAV of the accumulated gradients evaluated on a batch of inputs.
- **Random Pruning** - weights are pruned uniformly at random across all layers to a chosen fraction.
- **$L_0$  norm MBP** [264] (L0-MBP) - uses non-negative stochastic gates that choose which weights are set to zero as a smooth approximation to the non-differentiable  $L_0$ -norm.
- **$L_1$  norm MBP** [234] - applies  $L_1$  weight regularization and uses layer-wise magnitude pruning.
- **Lookahead pruning (lookahead)** - prunes weight paths that have the smallest magnitude across blocks of layers, unlike MBP that does not consider neighboring layers.
- **Layer-Adaptive Magnitude Pruning (LAMP)** - automatically decides the pruning ratio of each layer.

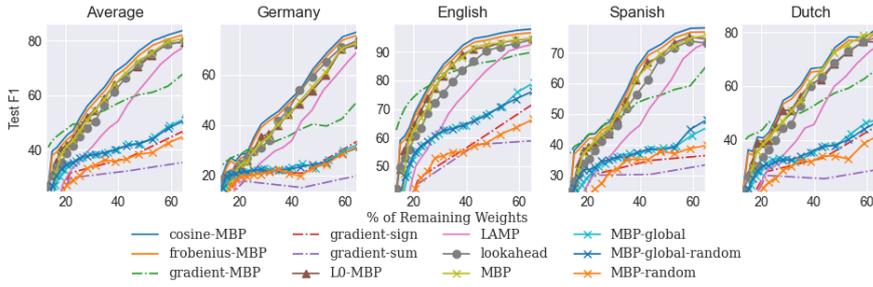


Figure 9.2: Zero-Shot Test F1 on Named Entity Recognition.

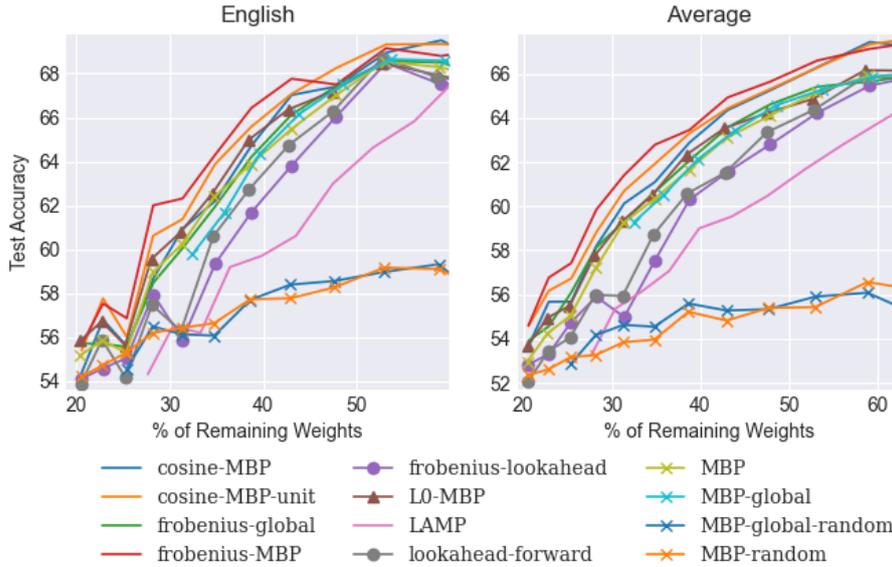


Figure 9.3: Test Accuracy on Question Answer Matching.

9.5 RESULTS

**NEWS CLASSIFICATION RESULTS** Figure 9.1 shows the results on news classification where a category for news article is predicted and evaluated in 5 languages and trained and iteratively pruned on English text. Firstly, it is observed that the trend in iterative pruning performance degradation is somewhat volatile. This is most observable for global random pruning (MBP-global-random) which for some higher compression rates performs better than previous less compressed states. From observations on preliminary experiments, news classification only requires 3 epochs to converge for standard fine-tuning on XLM-RoBERTa<sub>Base</sub>. It is posited that this task is relatively “similar” to the pretraining task and therefore able to easier recover from pruning steps.

**QUESTION ANSWER MATCHING** Figure 9.3 shows the supervised learning test accuracy on English and the zero-shot test accuracy on French and German for Question-Answer Matching (QAM). This in-

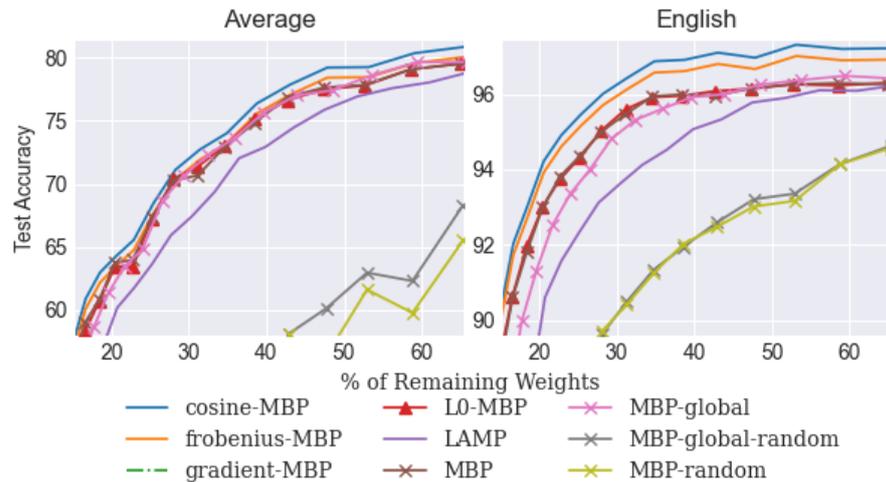


Figure 9.4: Part of Speech Tagging Test Accuracy.

involves predicting whether an answer answers a question correctly or not given a question-answer pair. Frobenius-MBP and Cosine-MBP are found to maintain higher accuracy across multiple pruning steps, outperforming baselines. More generally, we see there is close to 2% drop in average test accuracy drop in French and German when compared to testing on the samples from the same language used in training (English).

**NAMED ENTITY RECOGNITION** The Named Entity Recognition (NER) cross-lingual dataset is made up of CoNLL-2002 NER and CoNLL-2003 NER [371], covering English, Dutch, German and Spanish with 4 named entities. From Figure 9.2 we find that cross-lingual transfer of pruned models is most difficult in German and Dutch, which both come from the same language family, sharing commonalities such as word order and having similar vocabularies. The primary reason for the difficulty in maintaining performance in high compression rates for this NER dataset is that there is only 15k training samples, being significantly lower than the remaining XGLUE tasks (the majority contains 100k training samples). Thus, not only is there less training data to recover directly after each pruning step, but the pruning step interval itself is shorter. In contrast, English test performance is close to the original performance up until 25% of remaining weights, unlike the remaining languages. We find that gradient-MBP eventually overtakes MBP approaches past 20% remaining weights. However accuracy has reduced too much at this compression level. We find that Cosine-MBP and Frobenius-MBP weight regularizers achieve the best performing pruned model performance above 20% remaining weights, with Lookahead pruning and  $L_0$  regularized MBP being competitive in zero-shot performance.

**PART OF SPEECH TAGGING** The Part of Speech (PoS) tagging dataset consists of a subset of the Universal Dependencies treebank [313]

Prune Method	XNLI	NC	NER	PAWSX	POS	QAM	QADSM	WPR	Avg.
	73.48	80.10	82.60	89.24	80.34	68.56	68.06	73.32	76.96
Random	51.22	70.19	38.19	57.37	52.57	53.85	52.34	70.69	55.80
Global-Random	50.97	69.88	38.30	56.74	53.02	54.02	53.49	69.11	55.69
$L_0$ -MBP	64.75	78.98	56.22	72.09	71.38	59.31	53.35	71.70	65.97
$L_2$ -MBP	64.30	78.79	54.43	77.99	70.68	59.24	60.33	71.52	67.16
$L_2$ -Global-MBP	64.17	78.64	54.47	75.51	72.27	59.26	60.10	71.50	66.99
$L_2$ -Gradient-MBP	61.11	73.77	53.25	79.56	65.89	57.35	59.33	71.59	65.23
Lookahead	60.84	79.18	54.44	71.05	68.76	55.94	53.41	71.26	64.36
LAMP	58.04	63.64	51.92	66.05	67.43	55.36	52.42	71.09	60.74
Cosine-LMBP	65.01	79.01	55.18	78.27	71.60	57.27	61.32	71.33	<b>67.37</b>
Cosine-MBP	66.20	79.15	55.62	78.45	71.62	57.56	61.37	72.51	<b>67.81</b>
Frobenius-MBP	65.71	79.84	55.61	78.78	71.62	61.62	61.37	71.48	<b>68.25</b> <sup>†</sup>

Table 9.1: XGLUE Iterative Pruning @ 31% Remaining Weights of XLM- $R_{\text{Base}}$  - Zero Shot Cross-Lingual Performance Per Task and Overall Average Score (avg).

and covers 18 languages. In Figure 9.4, both Cosine-MBP and Frobenius-MBP tend to outperform baselines, although  $L_0$ -based pruning [264] has similar performance to Cosine-MBP for zero-shot accuracy. There is also a clear discrepancy between SSL accuracy (English) versus zero-shot accuracy (Average), the latter following closer to linear decay after 40-50% of weights remaining.

CROSS-LINGUAL NATURAL LANGUAGE INFERENCE Figure 9.5 shows the zero-shot cross-lingual transfer for various unstructured pruning methods. The accuracy on both the English test (i.e SSL generalization) and the average zero-shot test accuracy are consistently improved using cosine-MBP and frobenius-MBP, outperforming  $L_0$  pruning, Lookahead pruning and LAMP. Morphologically rich languages such as Arabic, Swahili and Turkish degrade in performance linearly once performance begins to drop after 60% of the remaining weights are pruned. This trend is roughly followed for all MBP-based pruning methods. Additionally, test accuracy on English can be maintained within 10% accuracy drop of the original test accuracy up to 20% of remaining weights for MBP, while Swahili can only be within a 10% accuracy drop up to 55% of the remaining weights. Hence, iterative pruning in the zero-shot setting leads to faster performance degradation for languages that are typologically or etymologically further from the language used for fine-tuning. When comparing English and the average zeros-shot test accuracy, the inflection point<sup>2</sup> and its slope is steeper for the average result for all pruning methods, not to mention the greater than 10% accuracy drop across pruning steps.

XGLUE AVERAGE RESULTS AND DISCUSSION Finally, the average task *understanding* score for our GradDrop variants and previous

<sup>2</sup> The point which the performance slope significantly steepens and drops are relatively large to previous pruning steps.

baselines is shown in Table 9.1. Both Cosine-MBP and Frobenius-MBP outperform MBP baselines, including LAMP and Lookahead pruning. Additionally, layer-wise pruning tends to outperform global pruning in this context. This can be explained by the clear discrepancy between weight norms of different layer types within each self-attention block. Global pruning chooses the majority of weights to prune from the layer type that has the smallest norm, leading to an information bottleneck, or layer collapse [227] for very high compression rates. This effect is due to layer normalization being applied after query, key and value (QKV) parameters, rescaling features such that weight magnitudes remain low. This is reflected on the right-most plots of Figure 9.6 and Figure 9.7 that show the weight norm by layer type for each layer for weight-MBP and gradient-MBP respectively. In both cases, QKV weights are distinctly higher than the remaining fully-connected layers (attention output layer, intermediate position-wise feedforward layer and the blocks output layer), with the exception that the output attention layer norm becomes higher between layer 3-8. This is also reflected in the corresponding pruned network weight distributions on the left that shows an order of magnitude in the different between weight value ranges, weight MBP, a bimodal distribution is created due to removal of lowest absolute weight magnitudes.

Lastly, for the majority of tasks the rate of performance drop for zero-shot test performance occurs close to 30% of remaining weights. This is roughly consistent for all pruning methods. To see how this reflected in the representations, Figure 9.8 visualizes the class separability via a t-SNE plot of two principal components of the last hidden representation corresponding to the [CLS] token of an iteratively pruned XLM-R<sub>Base</sub> for PAWSX. Even from only two principal components of a single token input, there is a clear change in class separability from 31% to 28% remaining weights, reflecting the difficulty the classifier has in linearly separating both classes.

## 9.6 SUMMARY

This chapter analysed iterative pruning in the zero-shot setting where a pretrained masked language model uses self-supervised learning on text from various languages but can only use a single language for downstream task fine-tuning. The results suggest some languages degrade in iterative pruning performance faster than others for some tasks (NER and XNLI) and propose a weight regularizer that biases the iteratively pruned model towards learning weight distributions close to the cross-lingually aligned pretrained state. This improves over well-established weight regularization methods for magnitude-based pruning in both the standard supervised learning setting and the zero-shot setting.

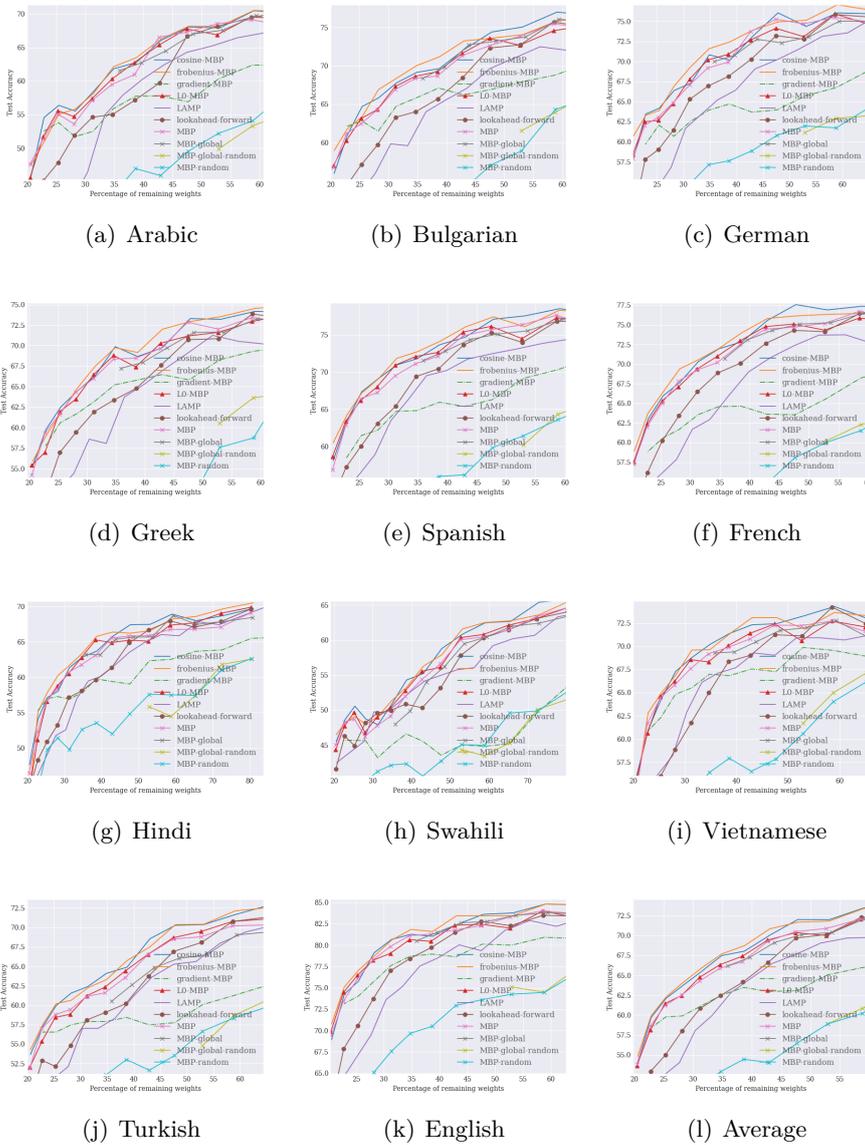


Figure 9.5: Zero-Shot XNLI Results Per Language After Iteratively Fine-Pruning XLM-RoBERTa<sub>Base</sub>

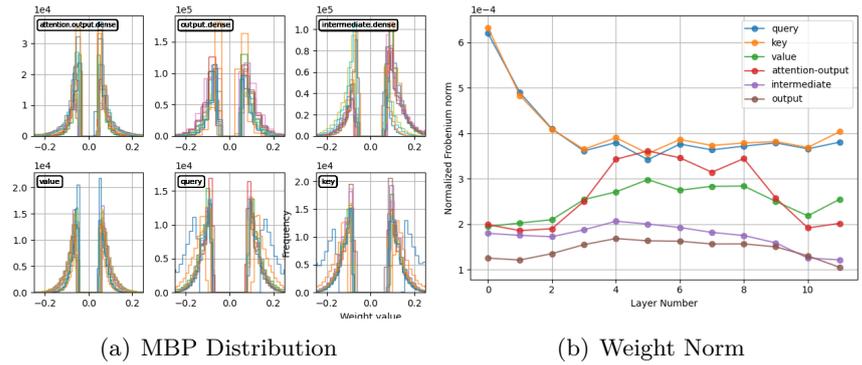


Figure 9.6: Weight Distribution and Weight Norms Per Layer of Remaining Weights After Iterative Pruning

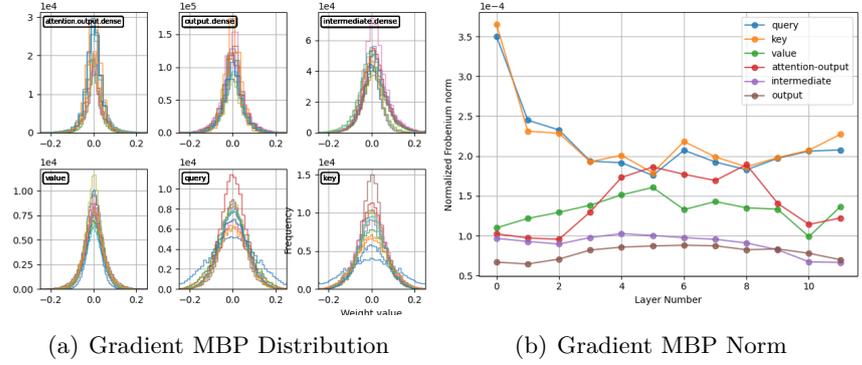


Figure 9.7: Gradient Distribution and Gradient Norms Per Layer of Remaining Weights After Iterative Pruning

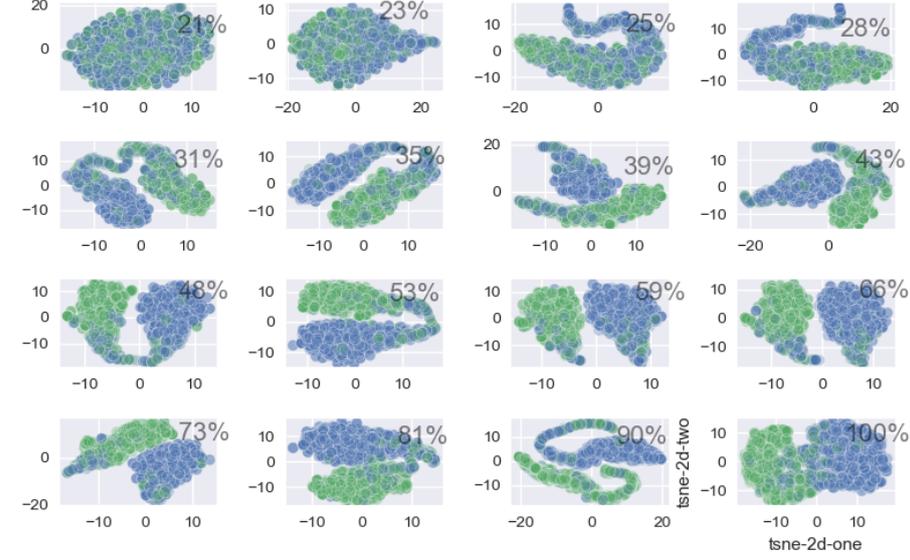


Figure 9.8: Class Separability Between Class Representations At Each Iterative Pruning Step on PAWSX.

---

## GRADIENT SPARSIFICATION FOR EFFICIENT FINE-TUNING OF TRANSFORMERS

---

Fine-tuning pretrained self-supervised language models is widely adopted for transfer learning to downstream tasks. Fine-tuning can be achieved by freezing gradients of the pretrained network and only updating gradients of a newly added classification layer, or performing gradient updates on all parameters. Gradual unfreezing [168] makes a tradeoff between the two by gradually unfreezing gradients of whole layers during training.

In this chapter, we propose to stochastically mask gradients to regularize pretrained language models for improving fine-tuning performance. We introduce *GradDrop* and variants thereof, a class of gradient sparsification methods that mask gradients in the backward pass. Unlike gradual unfreezing which is non-sparse and deterministic, GradDrop is sparse and stochastic. Experiments on the XGLUE benchmark [238] with XLM-R<sub>Large</sub> [71] show that *GradDrop* is competitive against methods that use additional translated data for intermediate pretraining and outperforms standard fine-tuning and gradual unfreezing.

### 10.1 INTRODUCTION

Fine-tuning pretrained transformer models for downstream tasks has been the defacto standard in natural language processing due to the recent successes of large-scale masked language modeling [69, 91, 216, 349]. This is usually achieved in one of two ways: (1) freeze the gradients of the pretrained portion of the network and perform SGD on a newly added task-specific layer/s or (2) perform SGD on both the pretrained and newly added layer/s. However, freezing all gradients of the pretrained layers can be too restrictive, particularly when the downstream task is dissimilar to the task of language modeling used during pretraining [337]. In contrast, unfreezing all layers may lead to negative transfer whereby irrelevant features are tuned for a downstream task or stability issues may arise when performing SGD for a large number of parameters [250].

Gradual unfreezing [169] is an alternative method that only tunes a subset of  $k$  layers and freezes the remaining layers for an epoch. In each successive epoch, the next subset of  $k$  layers are tuned and this

repeats in a top-down fashion (i.e unfreeze top  $k$  subset of layers to the bottom  $k$  subset of layers). Gradual unfreezing reduces training time by reducing the number of gradient updates after backpropagation. For the sole purpose of improving fine-tuning performance, gradual unfreezing could benefit from sparse gradient dropout alternatives that allow at least a subset of weights of all layers to be tuned at each epoch. Concretely, instead of restricting the freezing of gradients for *whole* layers, we can mask a percentage of gradients in *all* layers to allow the gradient to flow through the whole network.

Thus, in this chapter we propose *gradient dropout*, which we refer to as GradDrop. We find two particular variants of GradDrop significantly improve the fine-tuning of pretrained models, namely GradDrop-Epoch (weight masks are fixed over the whole epoch) and Layer-GradDrop (stochastically masks out gradients of whole layers). Our experiments focus on cross-lingual language models (LMs), namely XLM-R<sub>Large</sub> [68], given its wide adoption and success in transfer learning to various languages. In summary, our main contributions are as follows.

#### CONTRIBUTIONS

1. A dropout variant called *gradient dropout* (*GradDrop*) that regularizes fine-tuned models by randomly removing gradients during training. We also propose a variant (*GradDrop-Epoch*) that updates the gradient mask every epoch instead of every mini-batch.
2. Stochastic gradual unfreezing whereby layers are chosen at random for gradient updates at each epoch. We refer to this as Layer-GradDrop and compare this to standard fine-tuning and gradual unfreezing.
3. A comprehensive analysis of the how masking and fine-tuning can be used to improve cross-lingual transfer to downstream tasks without any *task-specific cross-lingual alignment* or *translate-train* training schemes.

## 10.2 RELATED RESEARCH

Before discussing our proposed regularizer, we review existing approaches to LM fine-tuning, cross-lingual LM fine-tuning and other methods that have explored masking strategies on pretrained LMs.

### 10.2.1 Language Model Fine-Tuning

**Adapters** have shown success by fine-tuning relatively small linear layers, referred to as *bottlenecks*, that are placed between pretrained frozen layers and generally only account for a small percentage (e.g 2-5%) of the overall number of parameters in the pretrained model. There are variants whereby some adapters are placed only on the

output of each self-attention block, within each self-attention block or combining adapters that have been independently trained for specific tasks and languages [340]. Current work predominantly focuses on training adapters for each task separately [167, 338, 339], which enables parallel training and subsequent combination of the weights. Rücklé et al. [366] remove adapters from lower layers during training and inference, incorporating structured dropout [104] with adapters [167]. This leads to a reduction in parameters while maintaining task performances and find further improvements when pruning adapters using Adapter Fusion [338].

**GRADUAL UNFREEZING** Howard and Ruder [169] gradually turning on gradients layer by layer for LM pretraining and fine-tuning, leading to a reduction in training time due to a reduction in gradient updates. Peters, Ruder, and Smith [337] have further explored which tasks benefit from fine-tuning when all gradients are active, when only the newly added fine-tuning layer gradients are active and when using gradual unfreezing. Their main finding is that when the underlying LM pretraining is semantically similar to the downstream task there is less need to deactivate gradients, while the semantically different tasks benefit more from activating all gradients for fine-tuning.

#### 10.2.1.1 *Cross-Lingual Fine-Tuning*

Ren et al. [356] use cross-lingual pretraining to improve performance on unsupervised neural machine translation (UNMT) by computing cross-lingual n-gram embedding and predicting an n-gram translation table from them. From this, they introduce cross-lingual MLM where they sampling n-grams for a given input text and predict the translation n-grams at each time step.

Muller et al. [302] show that multilingual BERT, a popular multilingual LM, can be viewed as the stacking of a multilingual encoder followed by a task-specific language-agnostic predictor. While the encoder is crucial for cross-lingual transfer and remains mostly unchanged during fine-tuning, the task predictor has little importance on the transfer and can be reinitialized during fine-tuning.

Eisenschlos et al. [101] perform multi-lingual LM fine-tuning (MultiFiT) by combining universal language model fine-tuning [169] with the quasi-recurrent neural network [43], subword tokenization [211] and use a cross-lingual LM teacher network to distill the monolingual fine-tuned model to the zero-shot setting. Fang et al. [107] improve cross-lingual fine-tuning by first performing cross-lingual alignment prior to downstream fine-tuning by first learning language independent representations which are then concatenated passed as input to another self-attention block that learns the cross-lingual features.

### 10.2.2 Language Model Fine-Masking

While standard LM fine-tuning remains the common approach to transfer learning in NLP, there has been other masking-related approaches. Zhao et al. [487] have learned a mask over the weights instead of fine-tuning the weights, showing this can lead to competitive performance to fine-tuning. In contrast to our work, we show that combining masking during fine-tuning is a preferred method for the same computational budget. Liu, Agarwal, and Venkataraman [256] use the change of the gradient magnitudes of a layer as a criterion to determine whether a layer is frozen. Hence, gradients that stagnate in a layer are most likely to be frozen during fine-tuning.

### 10.3 METHODOLOGY

In this section, we describe our main contribution, gradient dropout and variants thereof. We begin by first describing the self-attention blocks in transformers. Assume we have a sequence of vectors  $\mathbf{x}_1, \dots, \mathbf{x}_n$  where each vector  $\mathbf{x}_i \in \mathbb{R}^d$  (e.g  $d = 512$ ). We define  $\mathbf{Q} \in \mathbb{R}^{n \times d}$  to be a matrix representing the sequence where the  $i$ -th row of  $\mathbf{Q}$  corresponds to  $\mathbf{x}_i$ . The key  $\mathbf{K} \in \mathbb{R}^{d \times l}$ , value  $\mathbf{V} \in \mathbb{R}^{d \times l}$  and projection layer  $\mathbf{U} \in \mathbb{R}^{d \times o}$  parameters are defined where  $\mathbf{U}$  ensures the output dimensionality of the self-attention block is the same as the original input  $\mathbf{Q}$ . We can then define the self-attention as Equation 10.1,

$$\mathbf{Z} = \text{Softmax} \left( \frac{\mathbf{Q}\mathbf{K}}{\sqrt{dl}} \mathbf{V}^\top \mathbf{Q}^\top \right) \mathbf{Q}\mathbf{U} \quad (10.1)$$

where  $\mathbf{Q}\mathbf{U} \in \mathbb{R}^{n \times o}$  is matrix of new embeddings,  $\mathbf{Q}\mathbf{K}\mathbf{V}^\top \mathbf{Q}^\top \in \mathbb{R}^{n \times n}$  is a matrix representing the inner products in a new  $l$ -dimensional space and  $\text{Softmax}(\mathbf{Q}\mathbf{K}\mathbf{V}^\top \mathbf{Q}^\top)$  is a matrix where each row entry is positive and sums to 1. Note that *scaled* dot-product is used ( $1/\sqrt{dl}$ ) to avoid vanishing gradients of the Softmax, which may occur when  $dl$  is large.

The parameters for the  $j$ -th attention head  $\mathbf{K}^j, \mathbf{V}^j \in \mathbb{R}^{d \times l}$ ,  $\mathbf{U}^j \in \mathbb{R}^o$  for  $j = 1, \dots, n_a$  where  $n_a$  is the number of attention heads. Then we summarize the formulation of multi-headed self-attention as Equation 10.2

$$\begin{aligned} \mathbf{Z}^j &= \text{Softmax} \left( \frac{\mathbf{Q}\mathbf{K}^j}{\sqrt{dl}} (\mathbf{V}^j)^\top \mathbf{Q}^\top \right) \mathbf{Q}\mathbf{U}^j \\ \tilde{\mathbf{Z}} &= \text{Concat}(\mathbf{Z}^1, \dots, \mathbf{Z}^{n_a}) \\ \mathbf{Z} &= \text{Feedforward}(\text{LayerNorm}(\tilde{\mathbf{Z}} + \mathbf{Q})) \end{aligned} \quad (10.2)$$

where  $\mathbf{Z}^j \in \mathbb{R}^{n \times d_a}$  and  $\tilde{\mathbf{Z}} \in \mathbb{R}^{n \times d_a n_a}$  where  $d_a$  is the dimensionality of the self-attention output. The above formulation omits positional

embeddings which are learned embeddings that output representations that reflect sequential information in its inputs (i.e same token with different contexts won't have the same representations) by computing distance between token positions. Given this background, we now describe gradient dropout.

### 10.3.1 Gradient Dropout

After backpropagation, we apply a random binary mask on the gradients of  $\mathbf{K}$ ,  $\mathbf{V}$  and  $\mathbf{U}$ . For simplicity, let's assume  $\theta := \{\mathbf{K}, \mathbf{V}, \mathbf{U}\}$  and the gradients of  $\theta$  are represented as  $\mathbf{g} := \nabla_{\theta} \mathcal{L}^s(f_{\theta}(\mathbf{Q}), \mathbf{Y})$  where  $\mathbf{Y} \in \mathbb{N}^{n \times d}$  and represents one-hot targets of dimension  $d$ . A binary mask  $\mathbf{m}$  is then generated from a predefined distribution (e.g Bernoulli, Gaussian) and applied over the gradients. The gradient update rule with gradient dropout can then be expressed as,

$$\theta'_l = \theta_l - \alpha * \mathbf{g}_l \odot \mathbf{m}_l \quad (10.3)$$

where  $\alpha$  is the learning rate,  $\odot$  performs the Hadamard product (i.e the element-wise product of tensors) and  $l \in L$  is the layer index. Given that the stochastic noise induced by SGD through random mini-batch training regularizes DNNs, we too expect that the random dropping of gradients will have a similar regularization effect. Here,  $\mathbf{m}$  is generated from a Bernoulli distribution  $\mathbf{m} \sim \text{Bernoulli}(p)$ , where gradients are randomly zeroed with probability  $p$ .

Note that, different from Dropout [396] which randomly drops the intermediate activations in a supervised learning network under a single task setting, we perform the dropout on the gradient level. We focus on binary masks for  $\mathbf{m}$  as it is computationally efficient to generate and store low precision boolean tensors, in comparison to continuous noise (e.g drawn from a Gaussian distribution). Lastly, when applying gradient dropout layerwise (GradDrop-Epoch),  $\mathbf{m} \in \{0, 1\}^L$  where  $l$ -th element in  $\mathbf{m}$  corresponds to whether that layers gradients are activated or not. When  $\mathbf{m}_l = 1$ , a ones matrix  $\mathbf{1}$  of the same dimensionality as  $\mathbf{g}_l$  is applied, and zeros when  $\mathbf{m}_l = 0$ <sup>1</sup>.

### 10.3.2 Epoch-wise Gradient Dropout

We also propose a variant of GradDrop whereby the same dropout mask is applied to all mini-batches for a single epoch. The mask can be reset for successive epochs by uniformly sampling from the aforementioned Bernoulli distribution at the same dropout rate as before.

However, we also consider an accumulative mask whereby we sample from the Bernoulli distribution *without* replacement for each epoch

<sup>1</sup> Please see the supplementary material for a pseudocode example of GradDrop used with XLM-R.

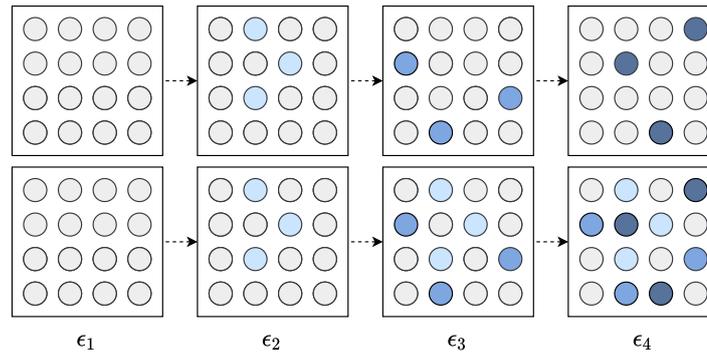


Figure 10.1: GradDrop-Epoch-Toggle (Top) and GradDrop-Epoch (Bottom). Grey represents frozen gradients, blue represents active gradients where darker blue indicates the recency of gradients turned on.

and this is the version we use for our experiments. Figure 10.1 shows the difference between GradDrop-Epoch when previous epoch masks are frozen once a new mask is applied (**GradDrop-Epoch-Toggle**) and when the previous epoch masks are left unfrozen (**GradDrop-Epoch**). In both cases, sampling without replacement is used, unlike standard GradDrop and like gradual unfreezing. This similarity to gradual unfreezing w.r.t. sampling without replacement aims to improve the stability during fine-tuning as only a subset of parameters are being updated for a whole epoch. Transformers are known to be difficult to train due to instability in optimization from their dependency on the residual branches within the self-attention blocks, as it amplifies parameter updates leading to larger changes to the model output [250]. These amplifications can be mitigated by stochastically freezing large portions of the network during fine-tuning when using GradDrop-Epoch, while allowing some gradient flow throughout all layers.

### 10.3.3 Annealed Variants of Gradient Dropout

Thus far we have assumed all gradient dropout variants (GDVs) have a fixed uniform gradient dropout rate throughout training. We can also apply each mask per minibatch or per epoch using a scheduled dropout rate that is non-uniform, such as an exponential decay or a linear decay. In this work, we focus on a linear schedule that begins at  $p = 0.9$ , reduces by  $p_\epsilon := p_{\epsilon-1} - 1/T$  at each epoch  $\epsilon$  until the last epoch  $T$  is reached where  $p = 0$ . In subsequent tables, models that have “+Anneal-” use this annealed GradDrop schedule.

### 10.3.4 Experimental Details

In our experiments, we focus on cross-lingual tasks from the XGLUE benchmark [238]. For all tasks, we only use English language training data for fine-tuning XLM-R<sub>Large</sub> and evaluate zero-shot test performance

on other multiple languages for each respective task. We do not use *any* cross-lingual alignment as a pretraining step and we also do not carry out the *translate-train* fine-tuning scheme (denoted with “-T” in subsequent tables), which first translates all languages to a well-resourced target language such as English and then fine-tunes on the downstream task. This is because our aim is to be competitive against cross-lingual alignment and translate-train based fine-tuning, as in many cases aligned or unaligned text is not easily available. For all GDVs, we apply gradient dropout to every layer apart from the input embedding layers and the task-specific classification layer that is on top of XLM-R<sub>Large</sub>. We ensure that there is no dropout used on the layers when using gradient dropout and when not using gradient dropout in standard fine-tuning, the dropout rate is set to the same rate when using gradient dropout.

Model	en	ar	bg	de	el	es	fr	hi	ru	sw	th	tr	ur	vi	zh	avg
<b>Original XLM-R</b>																
XLM-R <sub>Base</sub> Conneau et al.	84.6	78.4	78.9	76.8	75.9	77.3	75.4	73.2	71.5	75.4	72.5	74.9	71.1	65.2	66.5	74.5
XLM-R <sub>Large</sub> Conneau et al.	88.8	83.6	84.2	82.7	82.3	83.1	80.1	79.0	78.8	79.7	78.6	80.2	75.8	72.0	71.7	80.1
<b>FILTER</b>																
XLM-R <sub>Large</sub> Fang et al.	88.7	77.2	83.0	82.5	80.8	83.7	82.2	75.6	79.1	71.2	77.4	78.0	71.7	79.3	78.2	<u>79.2</u>
XLM-R <sub>Large</sub> ( <i>translate-train</i> )	88.6	82.2	85.2	84.5	84.5	85.7	84.2	80.8	81.8	77.0	80.2	82.1	77.7	82.6	82.7	82.6
FILTER	89.7	83.2	86.2	85.5	85.1	86.6	85.6	80.9	83.4	78.2	82.2	83.1	77.4	83.7	83.7	83.6
FILTER + Self-Teaching	89.5	83.6	86.4	85.6	85.4	86.6	85.7	81.1	83.7	78.7	81.7	83.2	79.1	83.9	83.8	83.9
<b>Ours</b>																
XLM-R <sub>Large</sub>	88.35	76.51	82.01	83.13	80.12	84.54	82.61	75.22	78.07	71.00	77.35	78.63	71.85	79.72	79.64	<u>79.25</u>
+GradFreeze-TopBottom	88.83	77.83	80.76	83.25	80.73	84.46	83.22	74.18	79.24	72.05	76.10	77.59	70.52	80.03	79.44	79.23
+GradFreeze-BottomUp	84.95	75.15	78.49	82.10	80.03	83.88	81.02	74.58	78.36	72.05	75.83	77.08	69.17	79.43	79.01	78.07
+GradDrop	<b>90.01</b>	<b>78.19</b>	<b>82.37</b>	<b>83.53</b>	<b>80.68</b>	<b>84.82</b>	<b>83.69</b>	<b>76.18</b>	<b>78.72</b>	<b>72.73</b>	<b>77.03</b>	<b>79.04</b>	<b>72.69</b>	<b>80.68</b>	<b>79.23</b>	<u>79.97</u>
+Anneal-GradDrop	88.49	76.88	82.81	83.54	80.13	85.07	82.90	78.11	78.14	71.04	76.72	78.39	72.47	80.17	79.28	79.58
+Layer-GradDrop	88.65	76.97	81.99	81.43	81.38	83.11	82.99	76.45	80.30	68.53	78.23	78.05	71.47	79.38	79.42	79.22
+Anneal-Layer-GradDrop	90.68	78.19	82.93	83.57	80.96	85.26	83.53	76.27	79.12	71.93	77.43	77.59	72.49	79.44	79.72	79.94
+GradDrop-Epoch	88.27	82.77	83.13	81.25	88.71	85.30	83.25	77.07	78.67	71.45	77.31	79.40	72.53	80.20	79.72	79.94

Table 10.1: XNLI zero-shot accuracy (apart from ‘en’) for each language. Results of fine-tuned XLM-R from prior work [69, 107] are from the XTREME benchmark [171].

#### 10.3.4.1 Baseline Masking Methods

Below we summarize the baselines considered in the proceeding experiments.

**GradFreeze+BottomTop** Howard and Ruder [168]: Gradually unfreezes gradients during training from the bottom layer to the top layer after each epoch.

**GradFreeze+TopBottom** Howard and Ruder [168]: Gradually unfreezes gradients during training from the top layer to the bottom layer after each epoch.

**Standard Fine-Tuning**: Fine-tunes the whole network on the downstream task.

**Unicoder** Huang et al. [175]: Unicoder is used to compare against a model which is trained with cross-lingual alignment using translation data.

**FILTER** Fang et al. [107]: As an upper bound on the expected perfor-

mance, we include FILTER which too uses but cross-lingual alignment, but is a larger model than Unicoder as it uses  $\text{XLM-R}_{\text{Large}}$ . FILTER, is currently [SoTA](#) on the XGLUE benchmark.

#### 10.3.4.2 *Our Gradient Masking Methods*

**GradDrop:** Randomly drops out gradients ( $p = 0.2$ ) on all layers for each minibatch.

**Layerwise GradDrop:** Randomly drops gradients ( $p = 0.2$ ) of a subset of layers for each mini-batch.

**Anneal GradDrop:** Randomly drops gradients of weights (Anneal-GradDrop) or a subset of layers (Anneal-Layer-GradDrop) for each mini-batch, starting at a high gradient dropout rate ( $p = 0.9$ ) and finishing low ( $p \approx 0$ ).

**GradDrop-Epoch:** Gradually unfreezes gradients randomly without replacement at each epoch until the whole network is unfrozen by the last epoch. Further comparisons between the GDVs are in the supplementary material.

### 10.4 RESULTS

In this section, we report the results of our proposed methods on the XGLUE benchmark tasks. We begin by discussing the zero-shot transfer results on sentence classification tasks.

#### 10.4.1 *Sentence Classification Results*

CROSS-LINGUAL NATURAL LANGUAGE INFERENCE (XNLI) [Table 10.1](#) shows the previous [SoTA](#) results on XNLI, our fine-tuned  $\text{XLM-R}_{\text{Large}}$ , GradFreeze (i.e gradual unfreezing), GradDrop and its variants. Standard GradDrop outperforms its other variants and all prior [SoTA](#) fine-tuning methods, including gradient freezing. We find a 0.72% increase in zero-shot test accuracy for GradDrop when compared to fine-tuning.

NEWS CLASSIFICATION [Table 10.2](#) shows the results on news classification where a category for news article is predicted and evaluated in 5 languages and trained on English. We find that both GradDrop and GradDrop-Epoch outperform the [SoTA](#) results (i.e FILTER) without any cross-lingual alignment techniques. We find that all GDVs outperform standard fine-tuning of  $\text{XLM-R}_{\text{Large}}$ . We also find that *gradual unfreezing* outperforms standard fine-tuning and best performance is obtained only after 3 epochs, which corresponds to only 6 of 24 layers being unfrozen. This suggests that the news classification task is closely aligned to the learned features in the pretrained  $\text{XLM-R}_{\text{Large}}$ . We also note that Layer-GradDrop *outperforms* FILTER by 1.26 percentage points.

	de	en	es	fr	ru	avg
FILTER Fang et al.	-	-	-	-	-	83.5
Unicoder Huang et al.	-	-	-	-	-	<u>83.5</u>
XLM-R <sub>Large</sub>	83.82	92.71	83.01	78.00	78.53	<u>83.21</u>
+ GradFreeze-TopDown	84.35	92.76	83.26	78.90	79.01	83.65
+ GradFreeze-BottomUp	79.75	90.27	81.31	75.18	75.56	80.41
+ GradDrop	84.64	92.84	83.26	78.57	79.30	83.41
+ Anneal Graddrop	74.36	93.13	78.49	81.49	81.87	81.87
+ Layer-GradDrop	<b>84.95</b>	<b>93.55</b>	<b>84.08</b>	<b>79.25</b>	<b>79.43</b>	<b><u>84.24</u></b> <sup>†</sup>
+Anneal Layer-GradDrop	82.77	92.62	82.70	77.38	80.78	83.25
+ GradDrop-Epoch	83.65	92.78	84.14	78.58	79.42	83.73

Table 10.2: Fine-Tuning XLM-R<sub>Large</sub> Results on News Classification. Test Accuracy on English and Zero-Shot Results for German, Spanish and French.

QUESTION ANSWERING MATCHING Table 10.3 shows the zero-shot test accuracy on English, French and German for the Question-Answer Matching (QAM) results. This involves predicting whether an answer answers a question correctly or not given a <question, answer> pair. We find that the GradDrop-Epoch variant outperforms other variants and improves significantly over standard fine-tuning by 3.16% and is only 0.42% below FILTER.

#### 10.4.2 Pairwise Classification

QUERY-AD MATCHING RESULTS In Query-Ad Matching (QADSM) task, we predict whether an advertisement is relevant to a query given an

	de	en	fr	avg
FILTER Fang et al.	-	-	-	73.4
Unicoder Huang et al.	-	-	-	<u>68.9</u>
XLM-R <sub>Large</sub>	70.10	70.83	68.52	<u>69.82</u>
+GradFreeze-TopDown	71.85	72.16	69.03	71.01
+GradFreeze-BottomUp	65.52	65.02	63.90	64.81
+GradDrop	72.14	72.53	70.49	71.72
+Anneal-GradDrop	72.02	72.19	70.17	71.46
+Layer-GradDrop	72.89	72.88	71.23	72.33
+Anneal Layer-GradDrop	72.79	73.07	71.05	72.31
+GradDrop-Epoch	<b>73.45</b>	<b>73.78</b>	<b>71.84</b>	<b><u>72.98</u></b> <sup>†</sup>

Table 10.3: Fine-Tuning XLM-R<sub>Large</sub> Zero-Shot Results on Question Answer Matching. Test Accuracy on German, English and French.

	de	en	fr	avg
FILTER Fang et al.	-	-	-	71.4
Unicoder Huang et al.	-	-	-	<u>68.4</u>
XLM-R <sub>Large</sub>	69.57	71.95	71.65	<u>71.05</u>
+GradFreeze-TopDown	69.03	71.85	72.16	71.01
+GradFreeze-BottomUp	66.02	69.37	70.28	68.56
+GradDrop	69.53	71.89	71.60	71.01
+Anneal-GradDrop	69.01	71.55	71.57	70.71
+Layer-GradDrop	<b>70.30</b>	<b>71.92</b>	<b>71.94</b>	<b>71.39<sup>†</sup></b>
+Anneal Layer-GradDrop	70.04	71.59	71.84	71.16
+GradDrop-Epoch	70.12	70.33	71.10	70.52

Table 10.4: Fine-Tuning XLM-R<sub>Large</sub> Results on Query-Ad Matching. Zero-shot Test Accuracy on German, English and French.

<query, advertisement> text input pair. We test performance on English and zero-shot test accuracy on French and German. From Table 10.4 we find that Layer-GradDrop outperforms the remaining GDVs, gradual unfreezing and is only 0.24% accuracy percentage points below FILTER.

CROSS-LINGUAL ADVERSARIAL PARAPHRASE IDENTIFICATION  
 The PAWS-X paraphrase identification dataset [461] consists of English, Spanish, French and German languages for evaluation. From Table 10.5 we find that Layer-GradDrop is the best performing GradDrop variant, improving performance by 3% and is competitive with FILTER i.e 1.57 F1 point difference.

	de	en	es	fr	avg
FILTER Fang et al.	-	-	-	-	93.8
Unicoder Huang et al.	-	-	-	-	<u>90.1</u>
XLM-R <sub>Large</sub>	85.23	93.65	88.70	89.35	<u>89.23</u>
+ GradFreeze-TopDown	85.13	93.15	87.03	88.15	88.33
+ GradFreeze-BottomUp	83.59	92.10	85.67	87.94	87.31
+ GradDrop	88.95	94.90	90.71	91.35	91.46
+ Anneal Graddrop	88.78	94.38	90.18	91.48	91.18
+ Layer-GradDrop	<b>88.9</b>	<b>95.72</b>	<b>91.05</b>	<b>91.98</b>	<b>92.23<sup>†</sup></b>
+ Anneal Layer-Graddrop	88.48	94.48	89.19	92.09	91.06
+ GradDrop-Epoch	88.7	94.55	90.55	90.80	91.15

Table 10.5: Fine-Tuning XLM-R<sub>Large</sub> Results on Cross-lingual Adversarial Paraphrase Identification. Test F1 score on English and Zero-Shot Results in German, Spanish and French.

	de	en	es	nl	avg
FILTER Fang et al.	-	-	-	-	82.6
Unicoder Huang et al.	-	-	-	-	<u>79.70</u>
XLM-R <sub>Large</sub> Conneau et al.	72.27	92.74	76.44	81.00	<u>80.61</u>
XLM-R <sub>Large</sub>	74.67	93.15	78.74	82.10	<u>82.16</u>
+ GradFreeze-TopDown	68.52	91.33	78.87	75.84	78.64
+ GradFreeze-BottomUp	65.05	89.42	75.11	75.84	76.35
+ GradDrop	74.20	91.23	76.22	79.18	80.21
+ Anneal GradDrop	74.36	93.13	78.49	81.49	81.87
+ Layer-GradDrop	74.36	93.13	78.49	81.41	81.85
+ Anneal Layer GradDrop	74.58	92.62	77.98	80.75	81.48
+ GradDrop-Epoch	<b>79.47</b>	<b>94.95</b>	<b>73.83</b>	<b>81.39</b>	<b>82.41<sup>†</sup></b>

Table 10.6: Fine-Tuning XLM-R<sub>Large</sub> Results on Named Entity Recognition. Test F1 score on English and Zero-Shot Results in German, Spanish and Dutch.

	en	ar	bg	de	el	es	fr	hi	it	nl	pl	pt	ru	th	tr	ur	vi	zh	avg
FILTER Fang et al.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	81.6
Unicoder [175]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	<u>79.6</u>
XLM-R <sub>Large</sub> (Ours)	96.37	69.66	89.77	91.92	87.99	89.49	90.70	71.86	93.06	88.91	84.83	90.47	86.55	57.44	72.71	64.09	57.82	63.27	<u>80.38</u>
+ GradFreeze-TopDown	96.59	66.58	87.64	90.53	87.49	80.17	77.48	70.15	88.12	88.00	83.43	87.09	85.19	55.63	72.47	65.34	59.06	56.88	77.66
+ GradFreeze-TopDown	96.80	65.33	88.40	90.40	89.63	81.41	83.25	70.12	90.16	88.18	83.09	87.19	85.20	55.73	72.26	66.96	58.50	57.85	78.11
+ GradFreeze-BottomUp	96.48	65.38	86.12	89.81	85.91	80.00	76.18	69.77	87.27	87.98	84.02	86.16	84.32	55.39	72.08	65.59	58.26	56.17	77.32
+ GradDrop	96.57	72.54	88.52	91.98	87.07	89.66	90.00	73.63	93.30	88.80	84.60	90.86	87.27	58.80	74.74	70.94	57.65	60.95	81.00
+ Anneal GradDrop	95.04	70.19	86.04	91.38	86.78	88.44	90.09	73.87	93.02	88.28	84.17	90.77	86.59	58.91	74.01	69.37	57.40	60.13	80.72
+ Layer-GradDrop	<b>97.01</b>	<b>74.12</b>	<b>88.65</b>	<b>91.67</b>	<b>86.38</b>	<b>89.56</b>	<b>90.10</b>	<b>74.82</b>	<b>93.28</b>	<b>89.03</b>	<b>84.74</b>	<b>90.81</b>	<b>87.33</b>	<b>59.70</b>	<b>75.15</b>	<b>70.89</b>	<b>58.10</b>	<b>64.29</b>	<b>81.42<sup>†</sup></b>
+ Anneal Layer-GradDrop	96.57	72.54	88.52	91.98	87.07	89.66	90.00	73.63	93.30	88.80	84.60	90.86	87.27	58.80	74.74	70.94	57.65	60.95	80.88
+ GradDrop-Epoch	97.03	72.21	88.55	91.87	86.57	88.73	89.57	74.43	92.84	89.00	84.36	90.73	86.69	58.34	75.46	69.32	57.58	54.91	80.45

Table 10.7: Fine-Tuning XLM-R<sub>Large</sub> Results on Part of Speech Tagging. Test F1 score on English and Zero-Shot Results in 17 other languages.

### 10.4.3 Structured Prediction Tasks

**NAMED ENTITY RECOGNITION** The Named Entity Recognition (NER) cross-lingual dataset is made up of CoNLL-2002 NER and CoNLL-2003 NER [371], covering English, Dutch, German and Spanish with 4 named entities. We find that GradDrop-Epoch outperforms standard fine-tuning, gradual unfreezing, the remaining GradDrop variants and it is 0.19% points from FILTER. GradDrop outperforms standard fine-tuning and is competitive with SoTA without any additional parameters or training data.

**PART OF SPEECH TAGGING** The Part of Speech (PoS) tagging dataset consists of a subset of the Universal Dependencies treebank [313] and covers 18 languages. From Table 10.7, we find that all our GradDrop variants outperform standard fine-tuning XLM-R<sub>Large</sub> and Layer-GradDrop is the best performing variant. Additionally, it is only 0.3% average test accuracy points away from FILTER, the method that uses additional cross-lingual alignment training and pseudo-label knowledge transfer. Again, GradDrop does not rely on language alignment and only uses English language training data. We find that the largest improvements are made on Arabic, Urdu and Turkish (which shares

	de	en	es	fr	it	pt	zh	avg
FILTER Fang et al.	-	-	-	-	-	-	-	74.7
Unicoder Huang et al.	-	-	-	-	-	-	-	<u>73.9</u>
XLM-R <sub>Large</sub>	76.91	77.78	75.67	74.60	68.18	77.53	62.58	<u>73.32</u>
+GradFreeze-TopDown	76.75	76.97	74.79	73.81	66.55	77.08	62.31	72.61
+GradFreeze-BottomUp	73.42	73.58	74.01	72.84	67.04	75.13	62.18	71.17
+GradDrop	77.43	77.74	75.76	74.52	68.52	77.77	62.60	73.44
+Anneal-GradDrop	77.02	77.56	75.15	74.83	68.91	76.98	62.44	73.27
+Layer-GradDrop	78.48	78.83	76.40	75.12	70.00	78.65	64.08	74.51
+Anneal Layer-GradDrop	78.00	78.41	76.32	75.36	69.29	78.74	63.67	74.25
+GradDrop-Epoch	<b>78.93</b>	<b>78.85</b>	<b>76.70</b>	<b>75.61</b>	<b>69.33</b>	<b>79.01</b>	<b>63.86</b>	<u>74.61</u> <sup>†</sup>

Table 10.8: Fine-Tuning XLM-R<sub>Large</sub> Results on Web Page Ranking. Normalized DCG on German, English, Spanish, French, Italian, Portuguese and Chinese.

	ar	de	en	es	hi	vi	zh	avg
XLM Lewis et al.	54.8	62.2	74.9	68.0	48.8	61.4	61.1	61.6
FILTER Fang et al.	-	-	-	-	-	-	-	74.7
Unicoder Huang et al.	-	-	-	-	-	-	-	66.0
XLM-R <sub>Large</sub>	64.11	72.17	85.13	70.83	60.73	71.52	71.81	<u>70.9</u>
+GradFreeze-TopDown	63.82	71.98	84.41	71.05	61.02	70.17	71.44	70.55
+GradFreeze-BottomUp	61.29	70.48	84.02	69.98	60.79	69.88	71.05	69.64
+GradDrop	64.91	72.66	85.47	71.00	60.98	71.90	72.12	71.29
+Anneal-GradDrop	64.74	72.53	85.29	70.89	61.05	71.71	72.22	71.20
+Layer-GradDrop	66.09	73.60	87.02	72.17	61.65	72.50	72.75	<b>72.25</b>
+Anneal Layer-GradDrop	65.01	72.66	85.47	71.03	61.22	71.85	72.56	71.40
+GradDrop-Epoch	65.60	73.19	86.32	71.87	61.58	72.29	72.59	72.01

Table 10.9: Cross-Lingual Transfer Results on MLQA. F1 on Arabic, German, English, Spanish, Hindi, Vietnamese and *Simplified* Chinese.

approximately 30% of its vocabulary with Arabic words written in Arabic).

#### 10.4.4 Sentence and Span Retrieval Tasks

WEB-PAGE RANKING aims to predict whether a web page is relevant (1-5 ratings, “bad” to “perfect”) to an input query and it is evaluated for 7 languages using the Normalized Discounted Cumulative Gain (nDCG). From Table 10.8, we see that GradDrop-Epoch is the best performing gradient dropout variant, with Layer-GradDrop being 0.1 nDCG points below Layer-GraDrop and standard fine-tuning being 1.29 points below GradDrop-Epoch. Moreover, GradDrop-Epoch is only 0.09 points from FILTER.

Models	Translation	#Params	XNLI	NC	NER	PAWSX	POS	QAM	QADSM	WPR	MLQA	avg
FILTER+Self-Teaching Fang et al.	Yes	550M	83.9	83.5	82.6	93.8	81.6	73.4	71.4	74.7	76.2	80.1
XL <sub>M</sub> -R <sub>Large</sub> -T Fang et al.	Yes	550M	82.6	-	-	-	-	-	-	-	-	-
Unicoder Huang et al.	No	255M	75.3	83.5	79.70	90.1	79.6	68.9	68.4	73.9	66.0	76.1
XL <sub>M</sub> -R <sub>Large</sub> Conneau et al.	No	550M	80.1	-	-	-	-	-	-	-	-	-
XL <sub>M</sub> -R <sub>Large</sub> Fang et al.	No	550M	79.2	83.2	-	-	-	-	-	-	-	-
XL <sub>M</sub> -R <sub>Large</sub> (Ours)	No	550M	79.25	83.21	80.61	89.23	80.38	69.82	71.05	73.27	70.21	77.45
+GradFreeze-TopDown	No	550M	79.23	83.65	78.64	88.33	78.11	71.01	71.01	72.61	70.55	77.02
+GradFreeze-BottomUp	No	550M	78.07	80.41	76.35	87.31	73.32	64.81	68.56	71.17	69.64	74.40
+GradDrop	No	550M	79.97	83.41	80.21	91.46	81.00	71.72	71.02	73.44	71.29	78.17
+Anneal-GradDrop	No	550M	79.58	81.87	81.87	91.18	80.72	71.46	70.71	73.27	71.20	77.98
+Layer-GradDrop	No	550M	<b>79.22</b>	<b>83.73</b>	<b>81.85</b>	<b>92.23</b>	<b>81.42</b>	<b>72.33</b>	<b>71.39</b>	<b>74.51</b>	<b>72.25</b>	<b>78.77</b> <sup>†</sup>
+Anneal-Layer-GradDrop	No	550M	79.94	84.24	81.48	91.06	80.88	72.31	71.16	74.25	71.40	78.52
+GradDrop-Epoch	No	550M	79.94	83.73	82.41	91.15	80.45	72.98	70.52	74.61	72.01	<b>78.64</b> <sup>††</sup>

Table 10.10: Zero Shot Cross-Lingual Performance Per Task and Overall Average Score (avg).

MULTILINGUAL QUESTION ANSWERING We use MLQA [232] for the a multilingual machine reading comprehension task, which contains QA annotations labeled in 7 languages, including English, Arabic, German, Spanish, Hindi, Vietnamese and Chinese. Again, we find that Layer-GradDrop and GradDrop-Epoch are the best performing GDVs. Layer-GradDrop increases F1 by 1.35 over standard fine-tuning, while being 2.35 below FILTER.

#### 10.4.5 An Analysis of Training Stability

We also analyse the stability of different GDVs, compared to standard fine-tuning and gradual unfreezing in Figure 10.2. In Figure 10.2(a), the best test performance is found after 3 epochs for all GD variants. On further inspection, fine-tuning with GradDrop-Epoch maintains test performance for further training epochs while standard fine-tuning decreases as the model begins to overfit. In the remaining 3 tasks (XNLI, POS and QAM) we see that GradDrop variants maintain a stable test performance over training epochs. By reducing the number of gradient updates it synthetically adds noise to the gradient signal, which can be helpful for tasks the learn quickly yet training stability is somewhat volatile, as in the new classification task.

#### 10.4.6 XGLUE Understanding Score

Finally, we show the average task *understanding* score for our GradDrop variants and previous baselines in Table 10.10. We find that GradDrop-Epoch and Layer-GradDrop are two methods which consistently outperform the remaining GradDrop variants, standard fine-tuning and in some cases, FILTER which uses translation data. To our knowledge, Layer-GradDrop sets a **SoTA** results on XGLUE for methods which *do not* use *translate-train* or translation language model *cross-lingual* alignment pretraining. Additionally, Layer-GradDrop is only 1.4 understanding score points from FILTER with their self-teaching loss.

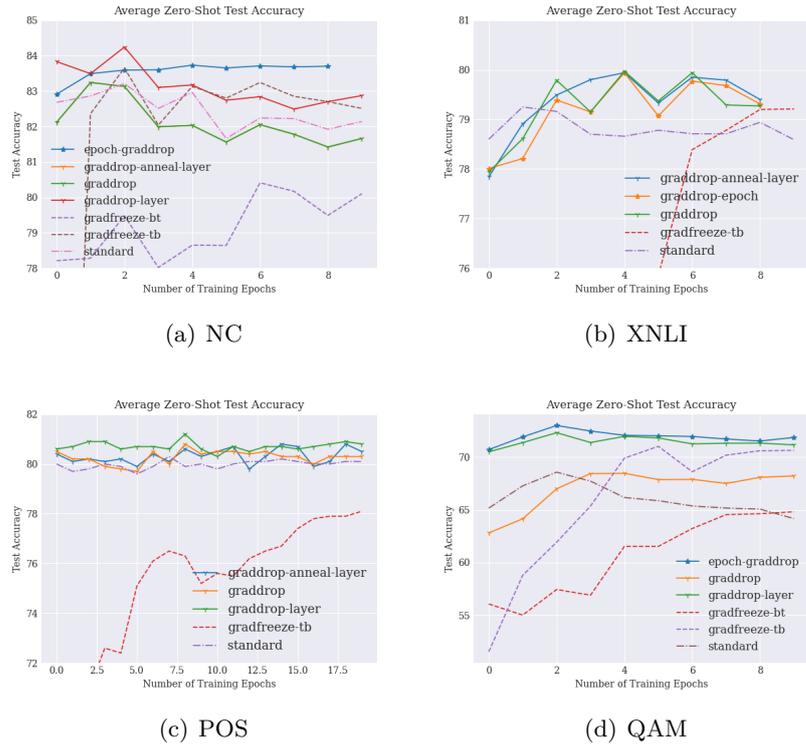


Figure 10.2: Test Performance Per Training Epoch.

#### 10.4.7 Per Language Analysis

Lastly, we inspect what languages do GDVs improve performance the most when compared to standard fine-tuning. We analyse the POS and XNLI task as they both include well-resourced and under-resourced languages in the evaluation set and are two different task types i.e structured prediction and sentence pair classification. Figure 10.3 shows how our best performing GDVs increase over standard fine-tuning and which languages we mostly attribute to the increase in average score. We find that there is significant performance improvements in Urdu, Turkish, Thai, Hindi and Arabic. In contrast, the performance of well-resourced languages such as English, Spanish, French, German and Italian are relatively minuscule. Likewise in XNLI, we find that biggest gains are made on Swahili and Arabic. We conclude, that GradDrop improves performance on under-resourced languages in particular. We posit that this may be because GradDrop forces the model to be robust to static gradients during training on English only, reducing the effects of overfitting to the English language.

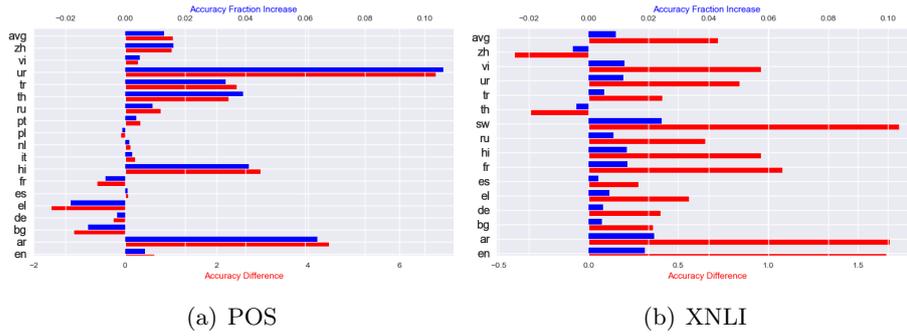


Figure 10.3: Test Performance Increase By Language in POS and XNLI. Red bars indicate accuracy increases or decreases, while blue indicates the fractional increase of GradDrop over standard fine-tuning.

## 10.5 SUMMARY

In this chapter, we proposed GradDrop and its multiple variants, showing that these variants can outperform standard fine-tuning of cross-lingual pretrained transformers. Specifically, epochwise- and layerwise- gradient dropout consistently outperform standard fine-tuning, gradual unfreezing and other gradient dropout variants. Additionally, it is competitive against [SoTA](#) methods that use translation data, cross-lingual alignment pretraining and self-distillation. We also find that gradient dropout significantly improves fine-tuning performance for under-resourced languages.



---

## SELF-DISTILLED PRUNING

---

As discussed, pruning aims to reduce the number of parameters while maintaining performance close to the original network. In this chapter, we propose a novel *self-distillation* based pruning strategy, whereby the representational similarity between the pruned and unpruned versions of the same network is maximized. Unlike previous approaches that treat distillation and pruning separately, we use distillation to inform the pruning criteria, without requiring a separate student network as in knowledge distillation. We show that the proposed *cross-correlation objective for self-distilled pruning* implicitly encourages sparse solutions, naturally complementing magnitude-based pruning criteria. Experiments on the GLUE [432] and XGLUE [238] benchmarks show that self-distilled pruning increases mono- and cross-lingual language model performance. Self-distilled pruned models also outperform smaller Transformers with an equal number of parameters and are competitive against (6 times) larger distilled networks. We also observe that self-distillation (1) maximizes class separability, (2) increases the signal-to-noise ratio, and (3) converges faster after pruning steps, providing further insights into why self-distilled pruning improves generalization.

### 11.1 INTRODUCTION

Neural network pruning [191, 299, 355] zeros out weights of a pretrained model with the aim of reducing parameter count and storage requirements, while maintaining performance close to the original model. The criteria to choose which weights to prune has been an active research area over the past three decades [11, 143, 191, 223, 294]. Lately, there has been a focus on pruning models in the transfer learning setting whereby a self-supervised pretrained model trained on a large amount of unlabelled data is fine-tuned to a downstream task while weights are simultaneously pruned. In this context, recent work proposes to learn important scores over weights with a continuous mask and prune away those that having the smallest scores [270, 374].

However, these learned masks double the number of parameters in the network, requiring twice the number of gradient updates to tune the original parameters *and* their continuous masks [374]. Ideally, we aim to perform task-dependent fine-pruning *without* adding more parameters to

the network, or at least far lesser than twice the count. More generally, we desire pruning methods that can recover from performance degradation directly after pruning steps, faster than current pruning methods while encoding task-dependent information into the pruning process. To this end, we hypothesize self-distillation may recover performance faster after consecutive pruning steps, which becomes more important with larger performance degradation at higher compression regime. Additionally, self-distillation has shown to encourage sparsity as the training error tends to 0 [292]. This implicit sparse regularization effect complements magnitude-based pruning criteria.

Hence, this chapter proposes to *maximize the cross-correlation* between output representations of the fine-tuned pretrained network and a pruned version of the same network – referred to as *self-distilled pruning* (SDP). Unlike typical knowledge distillation (KD) where the student is a separate network trained from random initialization, here the student is initially a masked version of the teacher. We focus on pruning fine-tuned monolingual *and* cross-lingual transformer models, namely BERT [90] and XLM-RoBERTa [68]. To our knowledge, this is the first study that introduces the concept of *self-distilled pruning* and analyze iterative pruning in cross-lingual contexts. In a nutshell, our main contributions are as follows:

1. We propose *self-distilled pruning*, a novel pruning framework that improves the generalization of pruned networks *without* introducing any additional parameters, using only a set of soft targets.
2. Inspired by the recent success of correlation-based objectives for representation learning in computer vision [481], we propose the use of a *cross-correlation objective for self-distillation pruning* that reduces redundancy and encourages sparse solutions, naturally fitting with magnitude-based pruning. This sets state of the art results for magnitude-based pruning.
3. We provide three insights as to why self-distillation leads to more generalizable pruned networks. Namely, we observe that self-distilled pruning (1) *recovers performance faster* after pruning steps (i.e., improves convergence), (2) *maximizes the signal-to-noise ratio* (SNR), where pruned weights are considered as noise, and (3) *improves the fidelity* between pruned and unpruned representations as measured by mutual information of the respective penultimate layers.
4. A comprehensive study of iterative pruning for monolingual and cross-lingual pretrained models on GLUE and XGLUE benchmarks. To our knowledge, this is the only work to include an evaluation of pruned model performance in the cross-lingual transfer setting.

## 11.2 BACKGROUND AND RELATED WORK

**Regularization-based pruning** can be achieved by using a weight regularizer that encourages network sparsity. Three well-established regularizers are  $L_1$ ,  $L_2$  and  $L_0$  weight regularization [257, 264, 465] for weight sparsity [143, 145]. For structured pruning, Group-wise Brain Damage [224] and SSL [440] propose to use Group LASSO [477] to prune whole structures (e.g., convolution blocks or blocks within standard linear layers). Park et al. [327] aim to avoid pruning small weights if they are connected to larger weights in consecutive layers and vice-versa, by constraining the Frobenius norm of the pruned layers to be close to unpruned network.

**Importance-based pruning** assigns a score for each weight in the network and removes weights with the lowest importance score. The simplest scoring criteria is magnitude-based pruning (MBP), which uses the lowest absolute value (LAV) as the criteria [143, 145, 355] or  $L_1/L_2$ -norm for structured pruning [257]. MBP can be seen as a zero-th order pruning criteria. However higher order pruning methods approximate the difference in pruned and unpruned model loss using a Taylor series expansion up until 1<sup>st</sup> order [147, 223] or the 2<sup>nd</sup> order, which requires approximating the Hessian matrix [274, 393, 433] for scalability. Lastly, the regularization-based pruning is commonly used with importance-based pruning e.g using  $L_2$  weight regularization alongside MBP.

**Knowledge Distillation** (KD) transfers the logits of an already trained network [162] and uses them as soft targets to optimize a student network. The student network is typically smaller than the teacher network and benefits from the additional information soft targets provide. There has been various extensions that involve distilling intermediate representations [360], distributions [176], maximizing mutual information between student and teacher representations [4], using pairwise interactions for improved KD [328] and contrastive representation distillation [308, 414].

**Self-Distillation** is a special case of KD whereby the student and teacher networks have the same capacity. Interestingly, self-distilled students often generalize better than the teacher [116, 460], however the mechanisms by which self-distillation leads to improved generalization remains somewhat unclear. Recent works have provided insightful observations of this phenomena. For example, [397] have shown that soft targets make optimization easier for the student when compared to the task-provided one-hot targets. [5] view self-distillation as implicitly combining ensemble learning and KD to explain the improvement in test accuracy when dealing with multi-view data. The core idea is that the self-distillation objective results in the network learning a unique set of features that are distinct from the original model, similar to features learned by combining the outputs of independent models in an ensemble.

Given this background on pruning and distillation, we now describe our proposed methodology *SDP*.

### 11.3 PROPOSED METHODOLOGY

We begin by defining a dataset  $\mathcal{D} := \{(X_i, y_i)\}_{i=1}^D$  with single samples  $s_i = (X_i, \mathbf{y}_i)$ , where each  $X_i$  (in the  $D$  training samples) consists of a sequence of vectors  $X_i := (\mathbf{x}_1, \dots, \mathbf{x}_N)$  and  $\mathbf{x}_i \in \mathbb{R}^d$ . For structured prediction (e.g., NER, POS)  $y_i \in \{0, 1\}^{N \times C}$ , and for single and pairwise sentence classification,  $y_i \in \{0, 1\}^C$ , where  $C$  is the number of classes. Let  $\mathbf{y}^S = f_\theta(X_i)$  be the output prediction ( $\mathbf{y}^S \in \mathbb{R}^C$ ) from the student  $f_\theta(\cdot)$  with pretrained parameters  $\theta := \{\mathbf{W}_l, \mathbf{b}_l\}_{l=1}^L$  for  $L$  layers. The input to each subsequent layer is denoted as  $\mathbf{z}_l \in \mathbb{R}^{n_l}$  where  $\mathbf{x} := \mathbf{z}_0$  for  $n_l$  number of units in layer  $l$  and the corresponding output activation as  $\mathbf{A}_l = g(\mathbf{z}_l)$ . The loss function for standard classification fine-tuning is defined as the cross  $\ell_{CE}(\mathbf{y}^S, \mathbf{y}) := -\frac{1}{C} \sum_{c=1}^C \mathbf{y}_c \log(\mathbf{y}_c^S)$ .

For self-distilled pruning, we also require an already fine-tuned teacher network  $f_\Theta$ , that has been tuned from the pretrained state  $f_\theta$ , to retrieve the soft teacher labels  $\mathbf{y}^T := f_\Theta(\mathbf{x})$ , where  $\mathbf{y}^T \in \mathbb{R}^C$  and  $\sum_c \mathbf{y}_c^T = 1$ . The soft label  $\mathbf{y}^T$  can be more informative than the one-hot targets  $\mathbf{y}$  used for standard classification as they implicitly approximate pairwise class similarities through logit probabilities. The Kullback-Leibler divergence  $\ell_{KLD}$  is then used with the main task cross-entropy loss  $\ell_{CE}$  to express  $\ell_{SDP-KLD}$  as shown in [Equation 11.1](#),

$$\ell_{SDP-KLD} = (1 - \alpha)\ell_{CE}(\mathbf{y}^S, \mathbf{y}) + \alpha\tau^2 D_{KLD}(\mathbf{y}^S, \mathbf{y}^T) \quad (11.1)$$

where  $D_{KLD}(\mathbf{y}^S, \mathbf{y}^T) = \mathbb{H}(\mathbf{y}^T) - \mathbf{y}^T \log(\mathbf{y}^S)$ ,  $\mathbb{H}(\mathbf{y}^T) = -\sum_c \mathbf{y}_c^T \log(\mathbf{y}_c^T)$  is the entropy of the teacher distribution and  $\tau$  is the softmax temperature. Following [\[162\]](#), the weighted sum of cross-entropy loss and KLD loss shown in [Equation 11.1](#) is used as our main SDP-based KD loss baseline, where  $\alpha \in [0, 1]$ . After each pruning step during iterative pruning, we aim to recover the immediate performance degradation by minimizing  $\ell_{SDP-KLD}$ . In our experiments, we use weight magnitude-based pruning as the criteria for SDP given MBP’s flexibility, scalability and miniscule computation overhead (only requires a binary tensor multiplication to be applied for each linear layer at each pruning step). However,  $D_{KLD}$  only distills the knowledge from the soft targets which may not propagate enough information about the intermediate dynamics of the teacher, nor does it penalize representational redundancy. This brings us to our proposed cross-correlation SDP objective.

#### 11.3.1 Maximizing Cross-Correlation Between Pruned and Unpruned Embeddings

Iterative pruning can be viewed as progressively adding noise  $\mathbf{M}_l \in \{0, 1\}^{n_{l-1} \times n_l}$  to the weights  $\mathbf{W}_l \in \mathbb{R}^{n_{l-1} \times n_l}$ . Thus, as the pruning steps

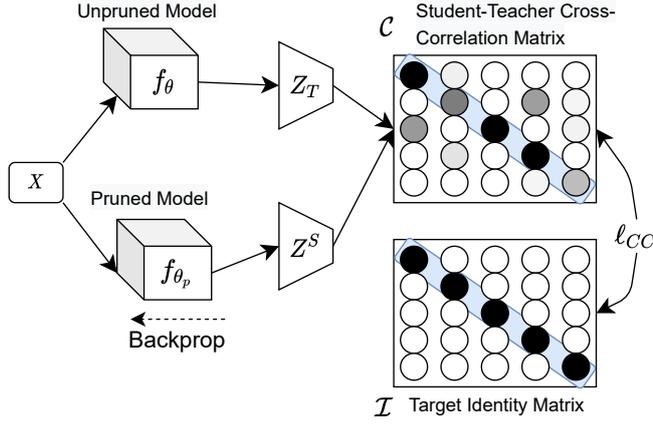


Figure 11.1: Self-Distilled Pruning w/ a Cross-Correlation Knowledge Distillation Loss.

increase, the outputs become noisier and the relationship between the inputs and outputs becomes weaker. Hence, a correlation measure is a natural choice for dealing with such pruning-induced noise. To this end, we use a cross-correlation loss to maximize the correlation between the output representations of the last hidden state of the pruned network and the unpruned network to reduce the effects of this pruning noise. The proposed *cross-correlation SDP loss function*,  $\ell_{CC}$ , is expressed in Equation 11.2, where  $\lambda$  controls the importance of minimizing the non-adjacent pairwise correlations between  $z^S$  and  $z^T$  in the correlation matrix  $\mathcal{C}$ . Here,  $m$  denotes the sample index in a mini-batch of  $M$  samples. Unlike  $\ell_{KLD}$ , this loss is applied to the outputs of the last hidden layer as opposed to the classification logit outputs. Thus, we have,

$$\ell_{CC} := \sum_i (1 - C_{ii})^2 + \lambda \sum_i \sum_{j \neq i} C_{ij}^2 \quad s.t.,$$

$$C_{ij} := \frac{\sum_m z_{m,i}^S z_{m,j}^T}{\sqrt{\sum_m (z_{m,i}^S)^2} \sqrt{\sum_m (z_{m,j}^T)^2}} \quad (11.2)$$

Maximizing correlation along the diagonal of  $\mathcal{C}$  makes the representations *invariant to pruning noise*, while minimizing the off-diagonal term *decorrelates the components* of the representations that are batch normalized. To reiterate,  $z^S$  is obtained from the pruned version of the network ( $f_{\Theta_p}$ ) and  $z^T$  is obtained from the unpruned version ( $f_{\Theta}$ ).

Since the learned output representations should be similar if their inputs are similar, we aim to address the problem where a correlation measure may produce representations that are instead *proportional* to their inputs. To address this, we use batch normalization across mini-batches to stabilize the optimization when using the cross-correlation loss, essentially avoiding local optima that correspond to degenerate representations that do not distinguish proportionality. In our experi-

ments, this is used with the classification loss and KLD distillation loss as shown in Equation 11.3.

$$\ell_{\text{SDP-CC}} = (1 - \alpha)\ell_{\text{CE}}(\mathbf{y}^S, \mathbf{y}) + \alpha^2\ell_{\text{KLD}}(\mathbf{y}^S, \mathbf{y}^T) + \beta\ell_{\text{CC}}(\mathbf{z}^S, \mathbf{z}^T) \quad (11.3)$$

To bring it all together, in Figure 11.1 we show the proposed framework of *Self-Distilled Pruning with cross-correlation loss* (SDP-CC), where  $\mathcal{I}$  is the identity matrix. Additionally, we provide a PyTorch based pseudo-code for SDP-CC in Figure 5 for a single epoch, in the general case. We see that the inner training loop requires little additional overhead, only requiring the extra computation (compared to normal pruning) to compute `distil_loss` and `cross_corr_loss`.

---

**Algorithm 5** PyTorch pseudo-code of SDP-CC.

---

```
# model: student transformer
# teacher_model: fine-tuned transformer
# alpha, beta: distillation loss weights
# method: chosen pruning criteria
# prune_rate: compression amount in [0, 1]
# lambda: weight on the off-diagonal terms
for x in loader: # batch loader of N samples
    # input_ids, attention_id & label_ids.
    x = tuple(xs.to(device) for xs in x)
    inputs = {"input_ids": x[0],
             "attention_mask": x[1], "labels": x[2]}
    outputs = student_model(**inputs)
    teacher_outs = teacher_model(**inputs)
    distil_loss = F.kld_divergence_loss(
        x[2], teacher_outs[2])
    # inputs last hidden representation for
    # [CLS] token and applies Equation 2.
    cross_corr_loss = cc_loss(outputs[1][-1],
                              teacher_outs[1][-1], lambda)
    # loss given as first element in tuple
    loss = outputs[0] + alpha * distil_loss
    + beta * cross_corr_loss
    # gradient clipping
    torch.nn.utils.clip_grad_norm_(
        model.parameters(), max_grad_norm)
    # compute backprop and update gradients
    loss.backward()
    optimizer.step()
# apply pruning after a whole epoch
with torch.no_grad():
    prune_method(model, method, prune_rate)
```

---

### 11.3.2 A Frobenius Distortion Perspective of Self-Distilled Pruning

To formalize the objective being minimized when using MBP with self-distillation, we take the view of *Frobenius distortion minimization* [FDM; 96] which says that layer-wise MBP is equivalent to minimizing the Frobenius distortions of a single layer. This can be described as  $\min_{\mathbf{M}: \|\mathbf{M}\|_0=p} \|\mathbf{W} - \mathbf{M} \odot \mathbf{W}\|_F$ , where  $\odot$  is the Hadamard product and  $p$  is a constraint of the number of weights to remove as a percentage of the total number of weights for a layer. Therefore, the output distortion is approximately the product of single layer Frobenius distortions. However, this minimization only defines a 1<sup>st</sup> order approximation of pruning

induced Frobenius distortions which is a loose approximation for deep networks. In contrast, the  $\mathbf{y}^T$  targets provide higher-order information outside of the  $l$ -th layer being pruned in this FDM framework because  $\Theta$  encodes information of all neighboring layers. Hence, we reformulate the FDM problem for SDP as an approximately higher-order MBP method as in Equation 11.4 where  $\mathbf{W}^T$  are the weights in  $f_\Theta$ .

$$\min_{\mathbf{M}: \|\mathbf{M}\|_0=p} \left[ \|\mathbf{W} - \mathbf{M} \odot \mathbf{W}\|_F + \lambda \|\mathbf{W}^T - \mathbf{M} \odot \mathbf{W}^T\|_F \right] \quad (11.4)$$

As described in [96, 147], the error function can be approximated with a Taylor Series (TS) expansion as  $\mathcal{E}(\mathbf{Z}_l^T) - \mathcal{E}(\mathbf{Z}_l) = \delta\mathcal{E}_l \approx \left(\frac{\partial\mathcal{E}_l}{\partial\mathbf{z}_l}\right)^\top \delta\mathbf{z}_l + \frac{1}{2}\delta\mathbf{z}_l^\top \mathbf{H}_l \delta\mathbf{z}_l + O(\|\delta\mathbf{z}_l\|^3)$ . Since, MBP corresponds to the 1<sup>st</sup> order term of this TS expansion, we can express the TS approximation of the error function  $\mathcal{E}(\cdot)$  with self-distilled pruning as shown in Equation 11.5, where  $\mathbf{Z}_l^*$  is the true latent representation of the input.

$$2\mathcal{E}(\mathbf{Z}_l) - \mathcal{E}(\mathbf{Z}_l^*) - \mathcal{E}(\mathbf{Z}_l^T) = \delta\mathcal{E}_l \approx \left(\frac{\partial\mathcal{E}_l}{\partial\mathbf{z}_l}\right)^\top \delta\mathbf{z}_l + \lambda \left(\frac{\partial\mathcal{E}_l^T}{\partial\mathbf{z}_l}\right)^\top \delta\mathbf{z}_l \quad (11.5)$$

### 11.3.3 How Does Self-Distillation Improve Pruned Model Generalization ?

We put forth the following insights as to the advantages provided by self-distillation for better model generalization, and later experimentally demonstrate their validity.

**Recovering Faster From Performance Degradation After Pruning Steps.** The first explanation for why self-distillation leads to better generalization in iterative pruning is that the soft targets bias the optimization and smoothen the loss surface through implicit similarities between the classes encoded in the logits. We posit this too holds true for performance recovery after pruning steps, as the classification boundaries become distorted due to the removal of weights. Faster convergence is particularly important for high compression rates where the performance drops become larger.

**Implicit Maximization of the Signal-to-Noise Ratio.** One explanation for faster convergence is that optimizing for soft targets translates to maximizing the margin of class boundaries given the implicit class similarities provided by teacher logits. Intuitively, task provided one-hot targets do not inform SGD of how similar incorrect predictions are to the correct class, whereas the teacher logits do, to the extent they have learned on the same task. To measure this, we use a formulation of the signal-to-noise ratio<sup>1</sup> (SNR) to measure the class separability and compactness differences between pruned model representations trained with and without self-distillation. We formulate

1 A measure typically used in signal processing to evaluate signal quality.

SNR as Equation 11.6, where for a batch of inputs  $\mathbf{X}$ , we obtain  $\mathbf{Z}$  output representations from the pruned network, which contain samples with  $C$  classes where each class has the same  $N$  number of samples. The numerator expresses the average  $\ell_2$  inter-class distance between instances of each class pair and the denominator expresses the intra-class distance between instances within the same class.

$$\text{SNR}(\mathbf{Z}) = \frac{1/N(C-1)^2 \sum_n^N \sum_{c=1}^C \sum_{i \neq c}^C \|\sqrt{\mathbf{Z}_{c,n}} - \sqrt{\mathbf{Z}_{i,n}}\|_2}{1/C(N-1)^2 \sum_{c=1}^C \sum_n^N \sum_{j \neq n} \|\sqrt{\mathbf{Z}_{c,n}} - \sqrt{\mathbf{Z}_{c,j}}\|_2} \quad (11.6)$$

This estimation is  $C - 1 \binom{C+1}{2}$  in the number of pairwise distances to be computed between the inter-class distances for the classes. For large output spaces (e.g., language modeling) we recommend defining the top  $k$ -NN classes for each class and estimate their distances on samples from them.

**Quantifying Fidelity Between Pruned Models Trained With and Without Self-Distillation.** A natural question to ask is *how much generalization power does the distilled soft targets provide when compared to the task provided one-hot targets?* If best generalization is achieved when  $\alpha = 1$  in Equation 11.1, this implies that the pruned network should have as high fidelity as possible with the unpruned network. However, as we will see there is a bias-variance trade-off between fidelity and generalization performance, i.e.,  $\alpha = 1$  is not optimal in most cases. To measure fidelity between SDP representations and standard fine-tuned representations, we compute their *mutual information* (MI) and compare this to the MI between representations of pruned models without self-distillation and standard fine-tuned models. The MI between continuous variables can be expressed as,

$$\begin{aligned} \hat{I}(\mathbf{Z}^T; \mathbf{Z}^S) &= H(\mathbf{Z}^T) - H(\mathbf{Z}^T | \mathbf{Z}^S) \\ &= -\mathbb{E}_{\mathbf{Z}^T} [\log p(\mathbf{Z}^T)] + \mathbb{E}_{\mathbf{Z}^T, \mathbf{Z}^S} [\log p(\mathbf{Z}^T | \mathbf{Z}^S)] \end{aligned} \quad (11.7)$$

where  $H(\mathbf{Z}^T)$  is the the entropy of the teacher representation and  $H(\mathbf{Z}^T | \mathbf{Z}^S)$  is the conditional entropy that is derived from the joint distribution  $p(\mathbf{Z}^T, \mathbf{Z}^S)$ . This can also be expressed as the KL divergence between the joint probabilities and product of marginals as  $I(Z^T; Z^S) = D_{\text{KLD}}[p(Z^S, Z^T) || p(Z^S)p(Z^T)]$ . However, these theoretical quantities have to be estimated from test sample representations. We use a  $k$ -NN based MI estimator [103, 207, 427, 428] which partitions the supports into a finite number of bins of equal size, forming a histogram that can be used to estimate  $\hat{I}(Z^S; Z^T)$  based on discrete counts in each bin. This MI estimator is given as,

$$I(z^S; z^T) \approx \epsilon \left( \log \frac{\phi_{[z^S]}(i, k_{[z^S]}) \phi_{[z^T]}(i, k_{[z^T]})}{\phi_z(i, k)} \right) \quad (11.8)$$

where  $\phi_{z^S}(i, k_{[z^S]})$  is the probability measure of the  $k$ -th nearest neighbour ball of  $z^S \in \mathbb{R}^{n_L}$  and  $\omega_{[z^T]}(i, k_{[z^T]})$  is the probability measure of

the  $k_y$ -th nearest neighbour ball of  $\mathbf{z}^T \in \mathbb{R}^{n_L}$  where  $n_L$  is the dimension of the penultimate layer. In our experiments, we use 256 bins for the histogram with Gaussian smoothing and  $k = 5$  (see [207] for further details).

#### 11.4 EXPERIMENTAL SETUP

**Datasets.** We perform experiments on monolingual tasks within the GLUE [432] benchmark<sup>2</sup> with pretrained  $\text{BERT}_{\text{Base}}$  and multilingual tasks from the XGLUE benchmark [238] with pretrained  $\text{XLMR}_{\text{Base}}$ . In total, this covers 18 different datasets, covering pairwise classification, sentence classification, structured prediction and question answering. To our knowledge, this work is the first to analyse iterative pruning in the context of cross-lingual models and their application on multilingual datasets.

**Iterative Pruning Baselines.** For XGLUE tasks, we perform 15 pruning steps on  $\text{XLM-RoBERTA}_{\text{Base}}$ , one per 15 epochs, while for the GLUE tasks, we perform 32 pruning steps on  $\text{BERT}_{\text{Base}}$ . The compression rate and number of pruning steps is higher for GLUE tasks compared to XGLUE, because GLUE tasks involve evaluation in the *supervised classification* setting; whereas in XGLUE we report in the more challenging *zero-shot cross-lingual transfer* setting with only a single language used for training (i.e., English). At each pruning step, we uniformly pruning 10% of the parameters for both the models. Although prior work suggests non-uniform pruning schedules (e.g., cubic schedule [494]), we did not see any major differences to uniform pruning. We compare the performance of the proposed SDP-CC method against the following baselines:

- **Random Pruning (MBP-Random)** - prunes weights uniformly at random across all layers. Random pruning can be considered as a lower bound on iterative pruning performance.
- **Layer-wise Magnitude Based Pruning (MBP)** - for each layer, prunes weights with LAV.
- **Global Magnitude Pruning (Global-MBP)** - prunes LAV weights anywhere in the network.
- **Layer-wise Gradient Magnitude Pruning (Gradient-MBP)** - for each layer, prunes the weights with LAV of the accumulated gradients evaluated on a batch of inputs.
- **1<sup>st</sup> Taylor Series Pruning (TS)** - prunes weights based on the LAV of  $|\text{gradient} \times \text{weight}|$ .
- **$L_0$  norm MBP [264]** - uses non-negative stochastic gates that choose which weights are set to zero as a smooth approximation to the non-differentiable  $L_0$ -norm.

<sup>2</sup> WNLI is excluded for known issues, see the Q. 12 on the [GLUE benchmark FAQ](#).

- **$L_1$  norm MBP** [234] - applies  $L_1$  weight regularization and uses MBP.
- **Lookahead pruning (LAP)** [327] - prunes weight paths that have the smallest magnitude across blocks of layers, unlike MBP that does not consider neighboring layers.
- **Layer-Adaptive MBP (LAMP)** [226] - adaptively compute the pruning ratio per layer.

For all above pruning methods we exclude weight pruning of the embeddings, layer normalization parameters and the last classification layer, as they play an important role for generalization and account for less than 1% of weights in both BERT and XLM-R<sub>Base</sub>.

- **Knowledge Distillation** – We also compare against a class of smaller knowledge distilled versions of BERT model with varying parameter sizes on the GLUE benchmark. We report prior results of *DistilBERT* [373] and also mini-BERT models including *TinyBERT* [186], *BERT-small* [420] and *BERT-medium* [420].
- **Self-Distilled Pruning Variant** – In addition, we consider maximizing the cosine similarity between pruned and unpruned representations in the SDP loss, as  $\ell_{\text{SDP-COS}} := \alpha \ell_{\text{CE}}(\mathbf{y}^S, \mathbf{y}) + \beta \left(1 - \frac{\mathbf{z}^S \cdot \mathbf{z}^T}{\|\mathbf{z}^S\| \|\mathbf{z}^T\|}\right)$ . Unlike cross-correlation, there is no decorrelation of non-adjacent features in both representations for SDP-COS. This helps identify whether the redundancy reduction in cross-correlation is beneficial compared to the correlation loss that does not directly optimize this.

**Hyperparameter Settings.** For **SDP-KLD**, we tested  $\alpha = [0.1, 0.2, 0.5, 0.8, 1]$  on three tasks for GLUE and XGLUE and extrapolate the best performing setting for the remaining tasks of both benchmarks with a fixed  $\tau = 0.9$ . We find  $\alpha = 0.5$  to perform the best in all cases. For **SDP-CC**, we perform tests with  $\beta = [10^{-6}, 2 \times 10^{-5}, 5 \times 10^{-5}, 10^{-4}]$ , finding that  $\beta = 2 \times 10^{-5}$  results in the best average performance. For SDP-COS, we set  $\beta = 0.05$ .

## 11.5 EMPIRICAL RESULTS

**PRUNING RESULTS ON GLUE.** Table 11.1 shows the test performance across all GLUE tasks of the different models with varying pruning ratios, up to *10% remaining weights* of original BERT<sub>Base</sub> along with mini-BERT models [373, 420] of varying size. However, for the CoLA dataset, we report at 20% pruning as nearly all compression methods have an MCC score of 0, making the compressed method performance indistinguishable. For this reason, the GLUE score (**Score**) is computed for all tasks and methods @10% apart from CoLA. The best performing compression method per task is marked in **bold**. We find that our proposed SDP approaches (all three variants) outperform against baseline pruning methods, with *SDP-CC* performing the best

Compression Method	Score (avg.)	Single Sentence		Similarity and Paraphrase			Natural Language Inference		
		CoLA (mcc)	SST-2 (acc)	MNLI (acc)	MRPC (f1/acc)	STS-B (pears./spear.)	QQP (f1/acc)	RTE (acc)	QNLI (acc)
BERT <sub>Base</sub> (Ours)	84.06	53.24	90.71	80.27	80.9/77.7	83.5/83.8	83.9/88.0	68.59	86.91
<b>Knowledge Distilled Baselines</b> (% parameters w.r.t. original BERT)									
DistilBERT (60%)	82.85	51.3	91.3	82.2	87.5/-.-	86.9/-.-	-./85.5	59.9	89.2
BERT-Medium (44.4%)	81.54	38.0	89.6	80.0	86.6/81.6	80.4/78.4	69.6/87.9	62.2	87.7
BERT-Small (20%)	79.02	27.8	89.7	77.6	83.4/76.2	78.8/77.0	68.1/87.0	61.8	86.4
BERT-Mini (10%)	76.97	0.0	85.9	75.1	74.8/74.3	75.4/73.3	66.4/86.2	57.9	84.1
BERT-Tiny (3.6%)	73.32	0.0	83.2	70.2	81.1/71.1	74.3/73.6	62.2/83.4	57.2	81.5
<b>Pruning Baselines</b>									
		20%	10%	10%	10%	10%	10%	10%	10%
Random	66.03	6.50	78.44	69.55	77.5/67.1	27.4/26.9	77.07/81.86	52.70	74.66
$L_0$ -MBP	77.25	31.68	83.37	75.61	78.4/68.2	75.9/75.7	81.56/86.49	<b>64.26</b>	82.62
$L_2$ -MBP	76.48	29.51	83.37	76.19	78.4/68.2	75.3/75.6	77.50/82.98	62.09	82.61
$L_2$ -Global-MBP	77.16	29.25	82.83	76.40	81.2/69.9	75.1/75.5	82.77/86.70	62.01	82.24
$L_2$ -Gradient-MBP	74.84	15.46	82.91	72.51	81.0/73.7	73.8/73.6	80.41/85.19	56.31	79.33
1 <sup>st</sup> -order Taylor	76.31	28.88	83.26	74.64	<b>83.0/74.8</b>	76.7/76.6	80.09/85.29	57.76	81.20
Lookahead	76.40	28.15	82.80	75.31	79.8/70.5	71.9/71.9	81.84/86.53	60.29	81.80
LAMP	74.03	20.31	83.26	74.27	72.3/63.7	73.7/74.1	79.32/85.07	58.84	81.09
<b>Proposed Methodology</b>									
$L_2$ -MBP + SDP-COS	77.83	31.80	86.00	75.68	81.6/72.2	76.4/76.3	81.39/86.68	61.73	83.07
$L_2$ -MBP + SDP-KLD	78.34	36.74	<b>87.96</b>	77.94	80.5/68.2	77.1/77.3	83.21/85.58	63.18	83.54
$L_2$ -MBP + SDP-CC	<b>78.90</b>	<b>36.77</b>	87.84	<b>78.04</b>	81.1/71.0	<b>77.3/77.5</b>	<b>83.79/86.37</b>	62.64	<b>84.20</b>

BERT- results reported from Jiao et al. [186], Sanh et al. [373], and Ture et al. [420] and MNLI results are for the matched dataset.

Table 11.1: GLUE benchmark results for pruned models @10% (or @20%) remaining weights.

across all tasks. We note that for the tasks with fewer training samples (e.g., CoLA has 8.5k samples, STS-B has 7k samples and RTE has 3k samples), the performance gap is larger compared to BERT<sub>Base</sub>, as the pruning step interval is shorter and less training data allows lesser time for the model to recover from pruning losses and also less data for teacher model to distil in the case of using SDP.

Smaller dense versions of BERT require more labelled data in order to compete with unstructured MBP and higher-order pruning methods such as 1<sup>st</sup> order Taylor series and Lookahead pruning. For example, we see BERT-Mini (@10%) shows competitive test accuracy with our proposed SDP-CC on QNLI, MNLI and QQP, the three datasets with the most training samples (105k, 393k and 364k respectively). Overall,  $L_2$ -MBP + SDP-CC achieves the highest GLUE score for all models at 10% remaining weights when compared to BERT-Base parameter count. Moreover, we find that  $L_2$ -MBP + SDP-CC achieves best performance for 5 of the 8 tasks, with 1 of the remaining 3 being from  $L_2$ MBP+SDP-KLD. This suggests that redundancy reduction via a cross-correlation objective is useful for SDP and clearly improve over SDP-COS which does not minimize correlations between off-diagonal terms. Figure 11.2 shows the performance across all pruning steps. Interestingly, for QNLI we observe the performance notably improves between 30-70% for SDP-CC and SDP-KLD. For SST-2, we observe a significant gap between SDP-KLD and SDP-CC compared to the pruning baselines and smaller versions of BERT, while TinyBERT becomes competitive at extreme compression rates (<4%).

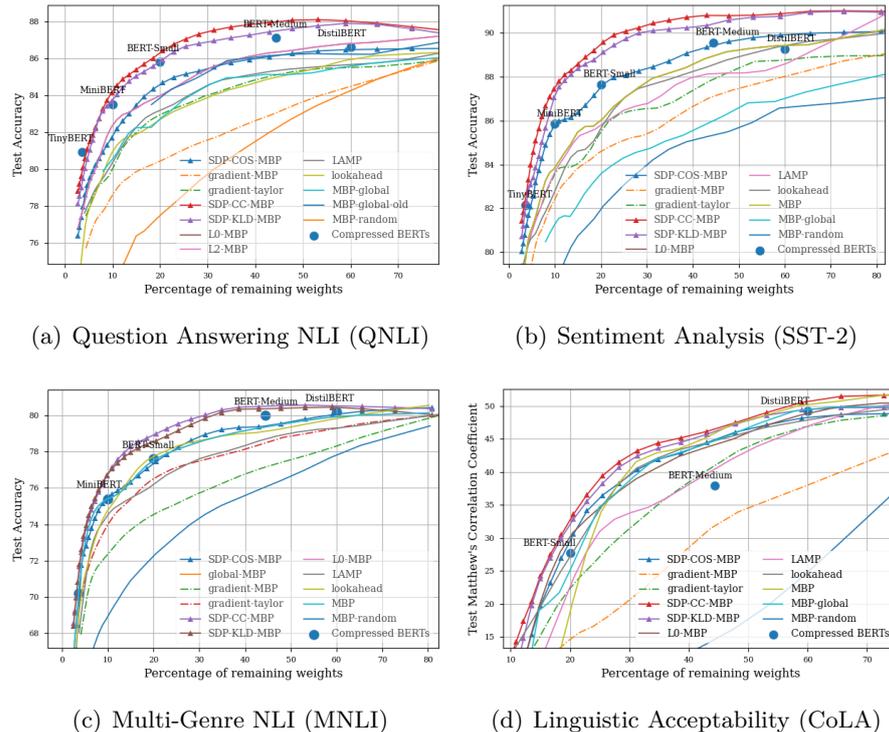
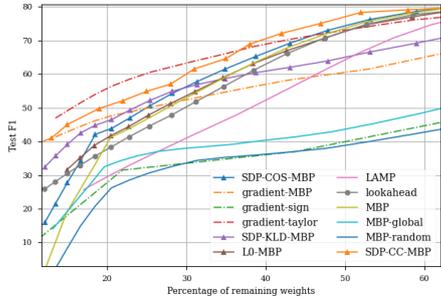


Figure 11.2: Iterative Pruning Test Performance on GLUE tasks.

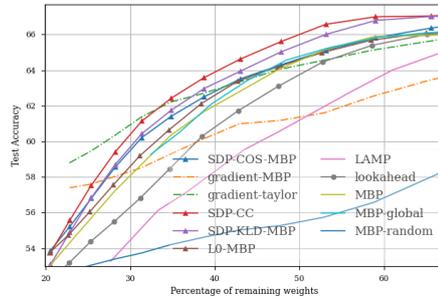
**Pruning Results on XGLUE.** We show the per task test performance and the *average task understanding* score on XGLUE for pruning baselines and our proposed SDP approaches in Table 11.2. Our proposed cross-correlation objective for SDP again achieves the best average (Avg.) score and achieves the best task performance in 6 out of 8 tasks, while standard SDP-KLD achieves best performance on one (news classification) of the remaining two. Most notably, we outperform methods which use higher order gradient information (1<sup>st</sup>-order Taylor) at 30% remaining weights, which tends to be a point at which XLM-R<sub>Base</sub> begins to degrade performance below 10% of the original fine-tuned test performance for SDP methods and competitive baselines.

In Figure 11.3, we can observe this trend from the various tasks within XGLUE. We note that the number of training samples used for retraining plays an important role in the rate of performance degradation. For example, of the 6 presented XGLUE tasks, NER has the lowest number of training samples (15k) of all XGLUE tasks and also degrades the fastest in performance (from 90% to 50% Test F1 at 30% remaining weights). In comparison, XNLI has the most training samples for retraining (433k) and maintains performance relatively well, keeping within 10% of the original fine-tuned model at 30% remaining weights.

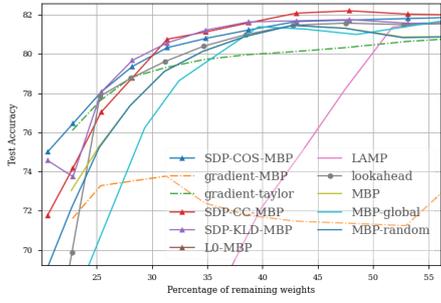
**SUMMARY OF RESULTS.** From our experiments on GLUE and XGLUE task, we find that SDP consistently outperforms pruning, KD



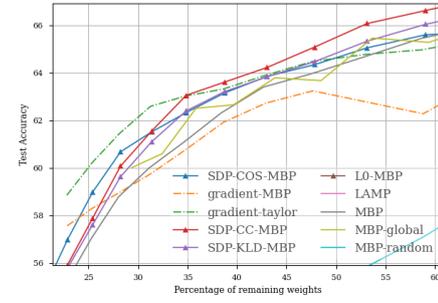
(a) Named Entity Recognition



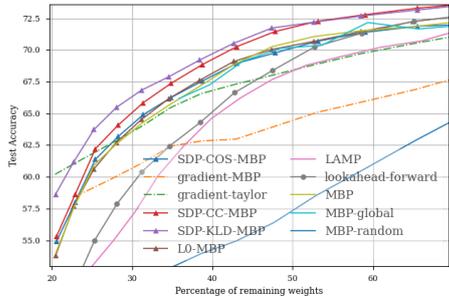
(b) Question Answer Matching



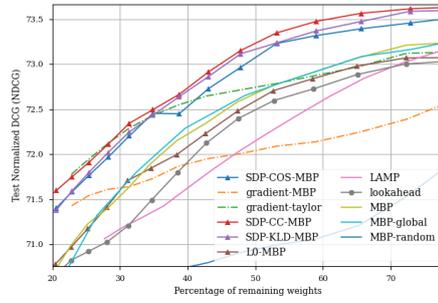
(c) News Classification



(d) Query Ad Matching



(e) XNLI



(f) WPR

Figure 11.3: Zero-Shot Results After Iteratively Fine-Pruning XLM-R<sub>Base</sub> on XGLUE tasks.

Prune Method	XNLI	NC	NER	PAWSX	POS	QAM	QADSM	WPR	Avg.
XLM-R <sub>Base</sub>	73.48	80.10	82.60	89.24	80.34	68.56	68.06	73.32	76.96
Random	51.22	70.19	38.19	57.37	52.57	53.85	52.34	70.69	55.80
Global-Random	50.97	69.88	38.30	56.74	53.02	54.02	53.49	69.11	55.69
$L_0$ -MBP	64.75	78.98	56.22	72.09	71.38	59.31	53.35	71.70	65.97
$L_2$ -MBP	64.30	78.79	54.43	77.99	70.68	59.24	60.33	71.52	67.16
$L_2$ -Global-MBP	65.12	78.64	54.47	79.13	71.37	59.26	60.61	71.80	67.55
$L_2$ -Gradient-MBP	61.11	73.77	53.25	79.56	65.89	57.35	59.33	71.59	65.23
1 <sup>st</sup> -order Taylor	64.26	79.34	63.60	<b>82.83</b>	68.94	61.69	62.42	72.28	69.09
Lookahead	60.84	79.18	54.44	71.05	68.76	55.94	53.41	71.26	64.36
LAMP	58.04	63.64	51.92	66.05	67.43	55.36	52.42	71.09	60.74
$L_2$ -MBP + SDP-COS	64.96	79.02	62.77	78.70	72.88	60.21	60.94	72.04	68.94
$L_2$ -MBP + SDP-KLD	65.94	<b>80.72</b>	64.50	79.25	73.18	61.66	61.09	71.84	<b>69.77</b>
$L_2$ -MBP + SDP-CC	<b>66.47</b>	79.73	<b>66.34</b>	80.03	<b>73.45</b>	<b>63.73</b>	<b>62.78</b>	<b>72.59</b>	<b>70.76</b>

Table 11.2: XGLUE Iterative Pruning @ 30% Remaining Weights of XLM-R<sub>base</sub> - Zero Shot Cross-Lingual Performance Per Task and Overall Average Score (Avg).

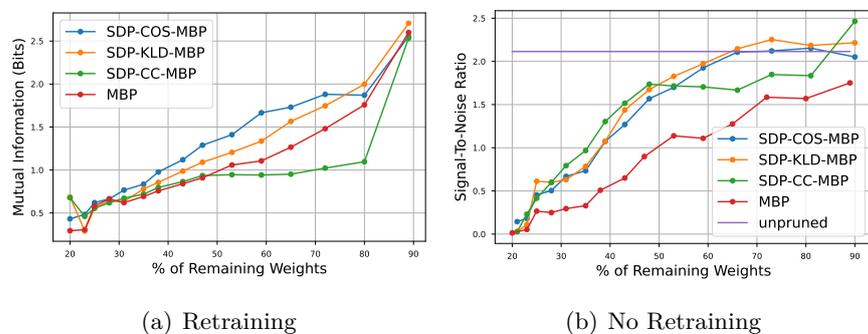


Figure 11.4: Mutual Information Between Unpruned and Pruned Representations (left) and Signal-To-Noise Ratio (right) from PAWS-X Development Set Representations.

and smaller BERT baselines. SDP-KLD and SDP-CC both outperform larger sized BERT models (BERT-Small), somewhat surprisingly, given that BERT-Small (and the remaining BERT models) have the advantage of large-scale self-supervised pretraining, while pruning only has supervision from the downstream task. For NER in XGLUE, higher order pruning methods such as Taylor-Series pruning have an advantage at high compression rates mainly due to lack of training samples (only 15k). Apart from this low training sample regime, SDP with MBP dominates at high compression rates both in standard and zero-shot settings.

MEASURING FIDELITY TO THE FINE-TUNED MODEL. We now analyse the empirical evidence that soft targets used in SDP may force higher fidelity with the representations of the fine-tuned model when compared to using MBP without self-distillation. As described in Section 11.3.3 we measure mutual dependencies between both repre-

presentations of models with the best performing hyperparameter settings of  $\alpha$ ,  $\beta$  and the softmax temperature  $\tau$ . We note that increasing the temperature  $\tau$  translates to “peakier” teacher logit distributions, encouraging SGD to learn a student with high fidelity to the teacher. From the LHS of Figure 11.4, we can see that SDP models have higher mutual information (MI) with the teacher compared to MBP, which performs worse for PAWS-X (similar on remaining tasks, not shown for brevity). In fact, the rank order of the best performing pruned models at each pruning step has a direct correlation with MI, e.g., SDP-COS-MBP maintains highest MI and the highest test accuracy for PAWS-X for the same  $\alpha$ . However, too high fidelity ( $\alpha = 1.$ ) led to worse generalization compared to a balance between the task provided targets and the teacher logits ( $\alpha = 0.5$ ).

**Self-Distilled Pruning Increases Class Separability and The Signal-To-Noise Ratio (SNR).** We also find that the SNR is increased at each pruning step as formulated in Section 11.3.3. From this observation, we find that *SDP-CC-MBP* using cross-correlation loss does particularly well in the 30%-50% remaining weights range. More generally, all 3 SDP losses clearly lead to better class separability and class compactness across all pruning steps compared to MBP (i.e., no self-distillation).

**Self-Distilled Pruning Recovers Faster Performance Degradation Directly After Pruning Steps.** Figure 11.5 shows how SDP with

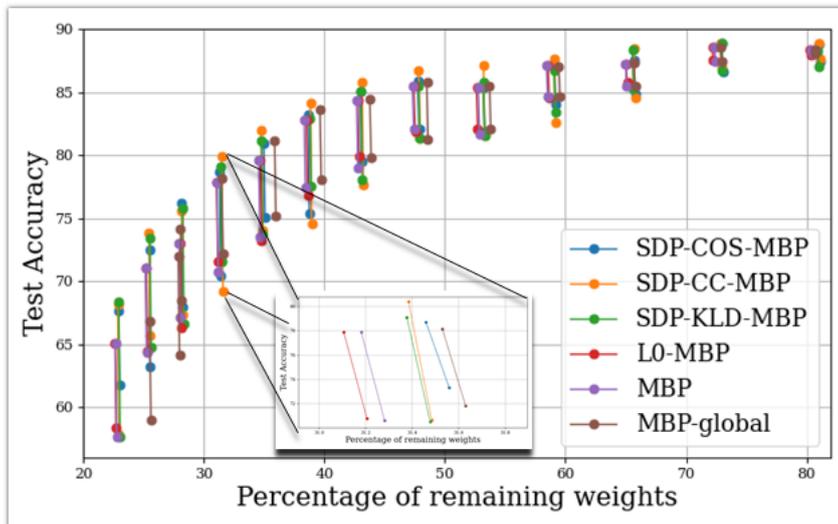


Figure 11.5: PAWS-X: Self-Distilled Pruning Leads to Better Performance Recovery.

Magnitude pruning (SDP-MBP) recovers during training in between pruning steps. The top of each vertical bar is the recovery development accuracy and the bottom is the initial performance degradation prior to retraining. We see that SDP pruned models degrade in performance more than magnitude pruning without self-distillation. This suggests

that SDP-MBP may force weights to be closer, as there is more initial performance degradation if weights are not driven to zero. However, the recovery is faster. This may be explained by recent work that suggests the stability generalization tradeoff [26].

## 11.6 SUMMARY

In this chapter, a novel *self-distillation* based pruning technique based on a *cross-correlation* objective is proposed. The confluence between pruning and self-distillation for masked language models was explored and its enhanced utility on downstream tasks in both monolingual and multi-lingual settings. The findings suggest that self-distillation aids in recovering directly after pruning in iterative magnitude-based pruning, increases representational fidelity with the unpruned model and implicitly maximize the signal-to-noise ratio. Additionally, the cross-correlation based self-distillation pruning objective minimizes neuronal redundancy and achieves state-of-the-art in magnitude-based pruning baselines, and even outperforms KD based smaller BERT models with more parameters. One research direction for the future is analysing the connection between self-distilled pruning and the lottery ticket hypothesis to clarify whether self-distillation may correspond to aiding in finding better pruning masks i.e lottery tickets.

This is the end of the proposed methods in this thesis and we now move to the conclusion of this thesis.

---

## CONCLUSION

---

This thesis has focused on improving training efficiency and model compression for DNNs in the context of [NLP](#) and [CV](#). In this pursuit, we have made contributions towards mitigating exposure bias, improving posterior approximation, knowledge distillation, pruning and dynamic weight sharing schemes for [CNN](#), [RNNs](#) and Transformer networks. This has ranged from compression methods that (1) are specifically for representing either inputs ([Chapter 5](#)) and outputs ([Chapter 4](#)) for classifiers, (2) distil semantic similarity to improve performance of both RL-based text generators ([Chapter 6](#)) and contrastively learned students for image classification ([Section 7.3.1](#)), (3) fuse weights of layers that have high alignment in [DNNs](#) ([Chapter 8](#)), (4) weight regularizers that improve zero-shot pruning performance ([Chapter 9](#)) and gradient-based pruning that stabilizes pretrained network fine-tuning ([Chapter 10](#)) and (5) an efficient and complementary framework for combining distillation and pruning ([Chapter 11](#)).

In the proceeding sections of this chapter, we conclude the work presented in this thesis. [Section 12.1](#) summarizes the work corresponding to each chapter. [Section 12.2](#) describes the main findings of each chapter and how it pertains to the overarching goal of improved model compression and training efficiency. Finally, [Section 12.3](#) discusses potential research avenues to further build upon the work on model compression described in this thesis.

### 12.1 THESIS SUMMARY

This thesis has proposed novel compression and training methods to address the aforementioned challenges and unexplored problems associated with compressing and retraining pretrained models. Below we provide a concise summary of each chapter.

- [Chapter 3 - Mitigating Exposure Bias](#). This chapter described the problem of reducing compounding errors in neural sequence predictors, namely conditional language modeling. Nearest-Neighbor Replacement Sampling was proposed to mitigate this problem by replacing a portion of input tokens with a sampled nearest neighbor of the past target with a truncated probability propor-

tional to the cosine similarity between the original word and its top- $k$  most similar words. This ensured that even if errors occur along the sequence that they are approximately bounded within a semantically similar neighboring token to the original token. This method improved perplexity of various language modeling datasets, is easy to implement and costs little to no additional computational overhead.

- **Chapter 4 - Model Compression via Error-Correcting Codes.** In this chapter, we discussed how softmax normalization in language models is a computational bottleneck given the high dimensional targets that are proportional to the size of vocabularies curated from modern corpora. Thus, we proposed to approximate the softmax function with error-correcting output codes. This requires less parameters than its softmax-based sequence modelling counterpart given sufficient separability between classes via error-checks. To achieve well-separated codes, a rank ordered codebook was proposed using pretrained embedding similarity where the number of error-correcting codes assigned to a token in the codebook is proportional to the cosine similarity between its word embedding and the most frequent token. We further show how this can also be used in conjunction with methods that mitigate exposure bias for further test time performance improvements, as discussed in [Chapter 3](#).
- **Chapter 5 - Model Compression Via Meta-Embedding.** In this chapter, we described our proposed method that reconstructs an ensemble of word embeddings as an auxiliary task that regularises a main task while both tasks share the learned meta-embedding layer. We carry out intrinsic evaluation (6 word similarity datasets and 3 analogy datasets) and extrinsic evaluation (4 downstream tasks). For intrinsic task evaluation, supervision comes from various labeled word similarity datasets. The experimental results show that the performance is improved for all word similarity datasets when compared to self-supervised learning methods with a mean increase of 11.33 in Spearman correlation. The proposed method shows the best performance in 4 out of 6 of word similarity datasets when using a cosine reconstruction loss and Brier’s word similarity loss. Moreover, improvements are also made when performing word meta-embedding reconstruction in sequence tagging and sentence meta-embedding for sentence classification.
- **Chapter 6 - Knowledge Distilled Reinforcement Learning.** In this chapter, we describe how policy-gradient based conditional text generators can benefit from being optimized for with semantic similarity based rewards, instead of sparse and sentence-length constrained n-gram measures such as BLEU and ROUGE. We proposed *Transferable Reward Learning* (TRL), a method that

incorporates model-based reward shaping to improve task-specific scores in relation to semantic similarity as a measure of language generation quality. We compared both unsupervised and supervised TRL models against ML training and previously proposed actor-critic models with model-free rewards such as BLEU and ROUGE-L. This combines transfer learning and knowledge distillation whereby the teacher is learned on a pairwise modeling task and then uses similarity scores between ground truth sequences and those generator from the student network.

- **Chapter 7 - Semantically Conditioned Negative Sampling.** In this chapter, we first proposed a class of semantic similarity based negative samplers that bias the sampling towards more informative negatives using a pretrained network. Namely, we propose Class and Instance Conditioned Negative Sampling. For class-wise negative sampling, we define the negative sampling distribution of each class by drawing negative samples proportional to the top- $k$  cosine similarity between pretrained word embeddings of the class labels. For instance-wise negative sampling, we proposed a top- $k$  instance-level similarity for defining the distribution by performing a forward pass with a pretrained network prior to training. Secondly, we proposed Contrastive Representation Mixup called *Latent Mixup* (LM), a variant of Mixup [484] that operates on latent representations between teacher positive and negative representations to produce harder pseudo negative sample representations that lie closer to the class boundaries. This is also carried out for the student network representations and a distance (or divergence) is minimized between both mixed representations. A theoretical analysis of conditioned sampling was carried out by reformulating the mutual information lower bound to account for semantic similarity of contrastive pairs, and describing the sample efficiency of semantically conditional negative sampling compared to uniform sampling. Our sample efficient contrastive learning models was compared to various knowledge distillation approaches for varying student and teacher network combinations. Prior work on metric learning-based KD had not yet explored defining non-uniform NS distributions which we argued is a critical element to the generalization performance and scalability of metric-learned neural networks. Lastly, we also include an ablation of loss functions for contrastive learning (CL) that were previously unexplored for KD, such as Centered Kernel Alignment and Pearson Correlation.
- **Chapter 8 - Layer Fusion for Deep Neural Networks.** In this chapter, we argued that too much information can be lost when zeroing out weights via pruning. Instead, we propose a dynamic weight sharing scheme whereby we fuse layers and describe various criteria for measuring layer similarity to rank layer pairs that are

subsequently used to fuse convolutional, fully-connected and attention layers. This leads to both computational and performance improvements over layer removal techniques, network pruning and shows competitive results compared to knowledge distillation. We report layer fusion using different fusion approaches: layer freezing, averaging and random *mixing*. Since aligning paths, layers and whole neural networks is non-trivial (neurons can be permuted and still exhibit similar behaviour), we also propose *alignment* measures for layer fusion. This includes (1) a sample-based Wasserstein distance metric to approximate the alignment cost between weight matrices and (2) the similarity between weight covariance matrices in fully-connected networks. We report the first results on using structured compression for large pretrained transformers and provide experimental results of different compression techniques with and without retraining. Thus, we identify the importance of retraining pretrained models for the aforementioned target tasks.

- **Chapter 9 - AlignReg: Weight Regularizers for Improved Pruning In The Zero-Shot Setting.** This chapter discussed the first analysis of pruning cross-lingual models and the first analysis of how various pruning criteria perform in the zero-shot setting, specifically for cross-lingual transfer. We proposed a weight regularizer that mitigates alignment distortion by minimizing the layer-wise Frobenius norm or unit similarity between the pruned model and unpruned model, avoiding overfitting to single language task fine-tuning. A post-analysis of weight distributions after pruning and how they differ across module types in Transformers.
- **Chapter 10 - Gradient Sparsification for Improving Fine-Tuning of Transformers.** In this chapter, we described our proposed dropout variant called *gradient dropout (GradDrop)* that regularizes fine-tuned models by randomly removing gradients during training. We also propose a variant (*GradDrop-Epoch*) that updates the gradient mask every epoch instead of every mini-batch. Stochastic gradual unfreezing whereby layers are chosen at random for gradient updates at each epoch. We referred to this as Layer-GradDrop and compared this to standard fine-tuning and gradual unfreezing. A comprehensive analysis of the masking and fine-tuning can be used to improve cross-lingual transfer to downstream tasks without any task-specific cross-lingual alignment or translate-train training schemes.
- **Chapter 11 - Self-Distilled Pruning.** In this chapter we introduced *self-distilled pruning*, a novel framework that combines pruning and knowledge distillation to improve the generalization of pruned networks *without* introducing any additional parameters, only a set of soft targets. Our secondary contribution was then a cross-correlation based objective that aims to remove representational redundancy, implicitly forcing redundant weights to zero

and thus, naturally fitting with magnitude-based pruning criteria. We find that self-distilled pruning with this cross-correlation objective achieves state of the art performance when compared to competitive baselines. Three insights are also provided to explain why self-distillation leads to more generalizable pruned networks. Namely, we discover that self-distilled pruning (1) recovers performance faster after pruning steps (i.e improves convergence), (2) maximizes the signal-to-noise (SNR) ratio, where pruned weights are considered noise and (3) improves the fidelity between pruned and unpruned representations as measured by mutual information of the respective penultimate layers. This was the first comprehensive study of iterative pruning for monolingual and cross-lingual pretrained models on GLUE and XGLUE benchmarks and the only work to include an evaluation of pruned model performance in the cross-lingual transfer setting.

## 12.2 CONTRIBUTIONS AND FINDINGS

In this section we provide a synopsis of the main contributions and findings of the work in this thesis. To contextualize the contributions and findings, we re-emphasize the research questions associated with each chapter and then finish with general findings that pertain to more than one chapter. We begin with [Chapter 3](#).

1. *Can a neural sequence predictor perform better at test time if we close the gap between training time and test time behavior through the use of neighborhood sampling to semantically bound the errors and mitigate the effects of exposure bias ?*

In [Chapter 3](#) we found that nearest-neighbor replacement sampling does indeed improve performance for text generation, particularly when given less training data, where the likelihood of compounding errors is increased. Constant and linear curriculum schedules show best performance for deciding the rate of NNRS throughout training. Our main finding is that NNRS should be considered as alternatives, or complementary to, other defacto standard exposure bias mitigation techniques such as scheduled sampling.

2. *Can language model training be made more efficient by approximating the softmax with error-correcting output codes, while maintaining close to performance when using the full softmax ?*

In [Chapter 4](#) we introduced a factored posterior approximation that multiplies probabilities corresponding to bits of error codes. Unlike, prior approximators such as the hierarchical softmax that use huffman codes, error codes have the flexibility to assign more

than one error code to a single token. This was useful because some tokens are harder to correctly classify than others and hence we can assign more error codes to these harder cases. Moreover, ordering the codebook such that semantically similar words have similar binary codes means that this ensures that incorrect predictions are at least semantically similar. From our results, we found that error-correcting output codes outperforms other softmax approximators, while reducing computation from  $\mathcal{O}(n^2)$  to  $\mathcal{O}(n \log n)$  where  $n$  is number of tokens in the vocabulary.

3. *Can we distil an ensemble of static word embeddings such that their single distilled representation is learned through supervised learning, unlike prior meta-embeddings that use self-supervised learning prior to fine-tuning ?*

We addressed the question whether outputs could be compressed into error codes, now move approximating the input representations. The main research question in this chapter was whether a meta-embedding could outperform single word embeddings, a concatenation of the ensemble and the remaining self-supervised meta-embedding approaches. We find that learning the meta-embedding as an auxiliary loss to the main task loss acts as a regularizer that improves generalization specifically for that task.

4. *Can we improve text generation by instead optimizing for sentence similarity between the predicted sequence and the ground truth sequence and does this improve text-generation based knowledge distillation ?*

In [Chapter 6](#) we found that using pretrained sentence similarity teacher networks to optimize policy-gradient based text generators can lead to generated sequences that are semantically diverse and lead to higher sentence similarity scores when compared to using more traditional word overlap measures which suffer from sparsity and impose undesired length constraints. This can be considered as a transfer-learning based knowledge distillation whereby the student text generator has learned from the pairwise teacher network on the different task of sentence similarity.

5. *Can pairwise-based knowledge distillation be improved w.r.t convergence time and generalization by biasing negative sampling towards harder negative samples ?*

In [Chapter 7](#) we find that contrastive-based knowledge distillation improves when you replace uniform sampling with our proposed semantically-conditioned negative sampling. In fact, when off the shelf pretrained models are available, this can also be used for standard supervised contrastive learning, not only for knowledge distillation. Moreover, not only does convergence time improve,

we also drastically reduce the number of negative samples required to reach the same performance as uniform sampling.

6. *For structured compression, can dynamically tying layers result in better performance than other methods such as tensor decomposition, knowledge distillation and structured pruning ?*

In [Chapter 8](#), we showed that tying similar layers can efficiently reduce network size while maintaining network density. We showed that this can outperform structured pruning for language modeling and image classification. Dynamic weight sharing is a novel compression method as weight sharing is usually carried out prior to training from random initialization. In this chapter, we identified that weights can be dynamically tied throughout retraining from a pretrained state to a relatively high compression ratio for structured compression.

7. *Does the choice of pruning criteria change in the zero-shot setting in comparison to standard supervised learning and can we improve pruning in the zero-shot setting by using alignment regularizers?*

We find that the results of pruned models using different pruning criteria in the standard supervised learning setting are also reflected in the zero-shot setting for cross-lingual models. However, we find that some languages are more prone to performance degradation, broadly speaking, these tend to be relatively under-resourced languages when compared languages such as English, Spanish, French, Russian and German. We also find that using a weight regularizer that constrains the fine-tuned model to preserve relative alignments between layers of the pretrained model (prior to fine-tuning) can mitigate performance drops on zero-shot languages. This is not found for  $\ell_1$  and  $\ell_2$  regularizers that are typically used with magnitude-based pruning. We conclude from this, that preserving angles between vectorized weight matrices to the original pretrained model is a useful regularization strategy for compressing Transformers in the zero-shot and few-shot setting.

8. *Can we improve fine-tuning of pretrained Transformers by stabilizing fine-tuning using gradient sparsification methods ?*

In [Chapter 10](#), we aimed to address this question by stochastically masking gradients to regularize pretrained language models for improving fine-tuning performance. Unlike dropout, the mask is not applied on weights or during the forward pass, only their gradients in the backward pass. We found that this two variants of this regularizer in particular improved task performance from the XGLUE benchmark [238] with XLM-R<sub>Large</sub> [71].

9. *Can pruning be improved by using self-distillation in conjunction and what distillation objective complements magnitude-based pruning ?*

In [Chapter 11](#) that using a self-distillation objective can further improve magnitude-based pruning and found that this is largely due to the objective implicitly maximizing the inter-class distance between samples and minimizing intra-class distance. We also found that our proposed cross-correlation self-distillation objective leads to best performance by reducing representational redundancy, which naturally fits well with weight magnitude-based pruning that can remove weights corresponding to units that are redundant.

### 12.3 FUTURE WORK

The machine learning subfield of model compression will remain a critical research area in the coming decades, as overparameterized networks continue to grow given the current trend of large scale pretraining. Below lists future research directions and then those directions that are potential continuations to the work described in this thesis.

#### 12.3.1 *Unify Compression Approaches*

As discussed in [Chapter 11](#), pruning and knowledge distillation are usually studied in isolation. This is also true for quantization and tensor decomposition. However, there is also room for combining these techniques such that they complement each other e.g low-bit precision training with pruning may be more beneficial than an equivalent compression rate only doing quantization or pruning. In future work, we aim to explore how pruning, knowledge distillation *and* quantization can be combined to achieve extreme compression levels. The seminal paper of Han, Mao, and Dally [143] that combines pruning, quantization and huffman coding has been directed towards this, however, the landscape of SoTA DNN models has changed since publication and benchmarks have become more challenging since then. Moreover, figuring out the optimal order of which compression techniques should be applied for a given set of tasks and architectures will be useful for both NLP and CV. A strong ablation study on many different architectures with various combinations and orders would be greatly insightful to the machine learning community.

#### 12.3.2 *More Compression Work on Large Non-Sparse Architectures*

The majority of the literature from previous works discussed in [Chapter 2](#) propose compression techniques in the context of CNNs since they have been used extensively over the past 3 decades, predominantly

for image-based tasks. This thesis has attempted to further work on larger Transformer architecture. Hence, one future direction is to extend existing techniques and analyse whether the empirical findings are similar on these more recent Transformer architectures and apply them to other important tasks (e.g text generation, speech recognition).

### 12.3.3 *Automatically Choosing Student Size in Model Distillation*

Current knowledge distillation approaches use fixed sized students during retraining. However, to get the desired tradeoff between performance versus student network size requires a manual iteration over a different student size in retraining. This is often used to visualize this tradeoff in papers, however automatically searching for student architecture during knowledge distillation is certainly an area of future research worth considering. In this context, meta learning and neural architecture search becomes important topics to bridge this gap between manually found student architectures to automatic techniques for finding the architectures. However, Neural Architecture Search must be efficient enough such that it does not significantly add to the retraining computational budget or storage. Meta learning [9, 378] has been successfully used for *learning to learn*. Meta-learning how these larger teacher networks learn could be beneficial for improving the performance and convergence of a distilled student network. To date, I believe this is an unexplored area of research.

### 12.3.4 *Bridging the Gap Between Data and Model Compression*

While neural network compression focuses on compressing the model, neural lossy data compression focuses on compressing data into a smaller number of bits by reconstructing the data and using the discretized encoded representation as input to arithmetic coding algorithm. The model used for reconstruction can be distilled from a larger model and the discretization process could be improved through the use of recent innovations for dynamic iterative quantization. Hence, not only is data being compressed, but the reconstruction step also used a compressed model, improving inference time for lossy compression algorithms.

### 12.3.5 *Further Theoretical Analysis*

Recent work has aided in our understanding of generalization in deep neural networks [28, 86, 305, 311, 377, 437] and proposed measures for tracking generalization performance while training DNNs. Further theoretical analysis of compression generalization is a worthwhile endeavour considering the growing importance and usage of compressing already trained neural networks. This is distinctly different than training models from random initialization and requires a new generalization

paradigm to understand how compression works for each type (i.e. pruning, quantization etc.).

Part I

APPENDIX



---

## BIBLIOGRAPHY

---

- [1] Prem Raj Adhikari and Jaakko Hollmen. “Multiresolution mixture modeling using merging of mixture components.” In: *Asian Conference on Machine Learning*. 2012, pp. 17–32.
- [2] Angeline Aguineldo, Ping-Yeh Chiang, Alex Gain, Ameya Patil, Kolten Pearson, and Soheil Feizi. “Compressing GANs using Knowledge Distillation.” In: *arXiv preprint arXiv:1902.00159* (2019).
- [3] Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc V Gool. “Soft-to-hard vector quantization for end-to-end learning compressible representations.” In: *Advances in Neural Information Processing Systems*. 2017, pp. 1141–1151.
- [4] Sungsoo Ahn, Shell Xu Hu, Andreas Damianou, Neil D Lawrence, and Zhenwen Dai. “Variational information distillation for knowledge transfer.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 9163–9171.
- [5] Zeyuan Allen-Zhu and Yuanzhi Li. “Towards Understanding Ensemble, Knowledge Distillation and Self-Distillation in Deep Learning.” In: *arXiv preprint arXiv:2012.09816* (2020).
- [6] Carl Allen and Timothy Hospedales. “Analogies Explained: Towards Understanding Word Embeddings.” In: *arXiv preprint arXiv:1901.09813* (2019).
- [7] Shun’ichi Amari et al. *The handbook of brain theory and neural networks*. MIT press, 2003.
- [8] Rie Kubota Ando and Tong Zhang. “A framework for learning predictive structures from multiple tasks and unlabeled data.” In: *Journal of Machine Learning Research* 6.Nov (2005), pp. 1817–1853.
- [9] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. “Learning to learn by gradient descent by gradient descent.” In: *Advances in Neural Information Processing Systems*. 2016, pp. 3981–3989.
- [10] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. “Large scale distributed neural network training through online distillation.” In: *arXiv preprint arXiv:1804.03235* (2018).

- [11] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. “Structured pruning of deep convolutional neural networks.” In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13.3 (2017), p. 32.
- [12] Arturo Argueta and David Chiang. “Accelerating Sparse Matrix Operations in Neural Networks on Graphics Processing Units.” In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 6215–6224.
- [13] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. “A latent variable model approach to pmi-based word embeddings.” In: *Transactions of the Association for Computational Linguistics* 4 (2016), pp. 385–399.
- [14] Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. “Fast and simple calculus on tensors in the log-Euclidean framework.” In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2005, pp. 115–122.
- [15] Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. “Geometric means in a novel vector space structure on symmetric positive-definite matrices.” In: *SIAM journal on matrix analysis and applications* 29.1 (2007), pp. 328–347.
- [16] Mikel Artetxe, Gorka Labaka, and Eneko Agirre. “Learning principled bilingual mappings of word embeddings while preserving monolingual invariance.” In: *Proceedings of the 2016 conference on empirical methods in natural language processing*. (2016).
- [17] Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. “N<sup>2</sup>n learning: Network to network compression via policy gradient reinforcement learning.” In: *arXiv preprint arXiv:1709.06030* (2017).
- [18] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. “Layer normalization.” In: *arXiv preprint arXiv:1607.06450* (2016).
- [19] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. “An actor-critic algorithm for sequence prediction.” In: *arXiv preprint arXiv:1607.07086* (2016).
- [20] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. “Deep equilibrium models.” In: *Advances in Neural Information Processing Systems*. 2019, pp. 688–699.
- [21] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. “Acicq: Analytical clipping for integer quantization of neural networks.” In: *openreview.net* (2018).
- [22] Cong Bao and Danushka Bollegala. “Learning Word Meta-Embeddings by Autoencoding.” In: *COLING* (2018).

- [23] Frédéric Barbaresco. “Information geometry of covariance matrix: Cartan-Siegel homogeneous bounded domains, Mostow/Berger fibration and Frechet median.” In: *Matrix information geometry*. Springer, 2013, pp. 199–255.
- [24] David Barber and Felix V Agakov. “The im algorithm: a variational approach to information maximization.” In: *Advances in neural information processing systems*. 2003, None.
- [25] Andrew G Barto, Richard S Sutton, and Charles W Anderson. “Neuronlike adaptive elements that can solve difficult learning control problems.” In: *IEEE transactions on systems, man, and cybernetics* 1.5 (1983), pp. 834–846.
- [26] Brian R Bartoldson, Ari S Morcos, Adrian Barbu, and Gordon Erlebacher. “The generalization-stability tradeoff in neural network pruning.” In: *arXiv preprint arXiv:1906.03728* (2019).
- [27] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. “Mutual information neural estimation.” In: *International Conference on Machine Learning*. 2018, pp. 531–540.
- [28] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. “Reconciling modern machine-learning practice and the classical bias–variance trade-off.” In: *Proceedings of the National Academy of Sciences* 116.32 (2019), pp. 15849–15854.
- [29] Mikhail Belkin, Daniel Hsu, and Ji Xu. “Two models of double descent for weak features.” In: *arXiv preprint arXiv:1903.07571* (2019).
- [30] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. “Deep rewiring: Training very sparse deep networks.” In: *arXiv preprint arXiv:1711.05136* (2017).
- [31] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. “Scheduled sampling for sequence prediction with recurrent neural networks.” In: *Advances in Neural Information Processing Systems*. 2015, pp. 1171–1179.
- [32] Yoshua Bengio, Nicholas Leonard, and Aaron Courville. “Estimating or propagating gradients through stochastic neurons for conditional computation.” In: *arXiv preprint arXiv:1308.3432* (2013).
- [33] Yoshua Bengio and Jean-Sébastien Senécal. “Adaptive importance sampling to accelerate training of a neural probabilistic language model.” In: *IEEE Transactions on Neural Networks* 19.4 (2008), pp. 713–722.
- [34] Yoshua Bengio, Jean-Sébastien Senécal, et al. “Quick Training of Probabilistic Neural Nets by Importance Sampling.” In: *AIS-TATS*. 2003, pp. 1–9.

- [35] Adam Berger. “Error-correcting output coding for text classification.” In: *IJCAI-99: Workshop on machine learning for information filtering*. 1999.
- [36] Rajendra Bhatia, Tanvi Jain, and Yongdo Lim. “On the Bures–Wasserstein distance between positive definite matrices.” In: *Expositiones Mathematicae* 37.2 (2019), pp. 165–191.
- [37] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. “Enriching word vectors with subword information.” In: *arXiv preprint arXiv:1607.04606* (2016).
- [38] Danushka Tarupathi Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. “Relational duality: Unsupervised extraction of semantic relations between entities on the web.” In: *Proceedings of the 19th international conference on World wide web*. 2010, pp. 151–160.
- [39] Danushka Bollegala and Cong Bao. “Learning word meta-embeddings by autoencoding.” In: *Proceedings of the 27th international conference on computational linguistics*. 2018, pp. 1650–1661.
- [40] Danushka Bollegala, Kohei Hayashi, and Ken-ichi Kawarabayashi. “Think Globally, Embed Locally—Locally Linear Meta-embedding of Words.” In: *arXiv preprint arXiv:1709.06671* (2017).
- [41] Danushka Bollegala, Kohei Hayashi, and Ken-ichi Kawarabayashi. “Think Globally, Embed Locally — Locally Linear Meta-embedding of Words.” In: *Proc. of IJCAI-EACL*. 2018.
- [42] Danushka Bollegala and James O’Neill. “A Survey on Word Meta-Embedding Learning.” In: *arXiv preprint arXiv:2204.11660* (2022).
- [43] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. “Quasi-recurrent neural networks.” In: *arXiv preprint arXiv:1611.01576* (2016).
- [44] Johanni Brea, Berfin Simsek, Bernd Illing, and Wulfram Gerstner. “Weight-space symmetry in deep networks gives rise to permutation saddles, connected by equal-loss valleys across the loss landscape.” In: *arXiv preprint arXiv:1907.02911* (2019).
- [45] Glenn W Brier. “Verification of forecasts expressed in terms of probability.” In: *Monthly weather review* 78.1 (1950), pp. 1–3.
- [46] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].
- [47] Elia Bruni, Gemma Boleda, Marco Baroni, and Nam-Khanh Tran. “Distributional semantics in technicolor.” In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*. Association for Computational Linguistics. 2012, pp. 136–145.

- [48] Maxime Bucher, Stéphane Herbin, and Frédéric Jurie. “Generating visual representations for zero-shot classification.” In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 2666–2673.
- [49] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. “Model compression.” In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 535–541.
- [50] Andres Buzo, A Gray, R Gray, and John Markel. “Speech coding based upon vector quantization.” In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.5 (1980), pp. 562–574.
- [51] Leopold Cambier, Anahita Bhiwandiwalla, Ting Gong, Mehran Nekuii, Oguz H Elibol, and Hanlin Tang. “Shifted and Squeezed 8-bit Floating Point format for Low-Precision Training of Deep Neural Networks.” In: *arXiv preprint arXiv:2001.05674* (2020).
- [52] Erick Cantu-Paz. “Pruning neural networks with distribution estimation algorithms.” In: *Genetic and Evolutionary Computation Conference*. Springer. 2003, pp. 790–800.
- [53] Kris Cao and Marek Rei. “A joint model for word embedding and word morphology.” In: *arXiv preprint arXiv:1606.02601* (2016).
- [54] Rich Caruana. “Multitask learning.” In: *Learning to learn*. Springer, 1998, pp. 95–133.
- [55] Carlos M Carvalho, Nicholas G Polson, and James G Scott. “The horseshoe estimator for sparse signals.” In: *Biometrika* 97.2 (2010), pp. 465–480.
- [56] Giovanna Castellano, Anna Maria Fanelli, and Marcello Pelillo. “An iterative pruning algorithm for feedforward neural networks.” In: *IEEE transactions on Neural networks* 8.3 (1997), pp. 519–531.
- [57] Yevgen Chebotar and Austin Waters. “Distilling Knowledge from Ensembles of Neural Networks for Speech Recognition.” In: *Interspeech*. 2016, pp. 3439–3443.
- [58] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. *Big Self-Supervised Models are Strong Semi-Supervised Learners*. 2020. arXiv: [2006.10029](https://arxiv.org/abs/2006.10029) [cs.LG].
- [59] Welin Chen, David Grangier, and Michael Auli. “Strategies for training large vocabulary neural language models.” In: *arXiv preprint arXiv:1512.04906* (2015).
- [60] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. “Compressing neural networks with the hashing trick.” In: *International conference on machine learning*. 2015, pp. 2285–2294.

- [61] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. “Improved baselines with momentum contrastive learning.” In: *arXiv preprint arXiv:2003.04297* (2020).
- [62] Xinlei Chen and C Lawrence Zitnick. “Mind’s eye: A recurrent visual representation for image caption generation.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 2422–2431.
- [63] Yen-Chun Chen, Zhe Gan, Yu Cheng, Jingzhou Liu, and Jingjing Liu. “Distilling the Knowledge of BERT for Text Generation.” In: *arXiv preprint arXiv:1911.03829* (2019).
- [64] Jang Hyun Cho and Bharath Hariharan. “On the efficacy of knowledge distillation.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 4794–4802.
- [65] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. “Pact: Parameterized clipping activation for quantized neural networks.” In: *arXiv preprint arXiv:1805.06085* (2018).
- [66] Joshua Coates and Danushka Bollegala. “Frustratingly Easy Meta-Embedding–Computing Meta-Embeddings by Averaging Source Word Embeddings.” In: *arXiv preprint arXiv:1804.05262* (2018).
- [67] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. “Natural language processing (almost) from scratch.” In: *Journal of Machine Learning Research* 12.Aug (2011), pp. 2493–2537.
- [68] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. “Un-supervised cross-lingual representation learning at scale.” In: *arXiv preprint arXiv:1911.02116* (2019).
- [69] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. “Un-supervised Cross-lingual Representation Learning at Scale.” In: *Proceedings of the 58th Conference of the Association for Computational Linguistics, ACL 2020, Virtual Conference, July 6-8, 2020*. 2020, pp. 8440–8451. URL: <http://arxiv.org/abs/1911.02116>.
- [70] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. “Supervised Learning of Universal Sentence Representations from Natural Language Inference Data.” In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Asso-

- ciation for Computational Linguistics, 2017, pp. 670–680. URL: <https://www.aclweb.org/anthology/D17-1070>.
- [71] Alexis Conneau and Guillaume Lample. “Cross-lingual Language Model Pretraining.” In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*. 2019, pp. 7057–7067. URL: <http://papers.nips.cc/paper/8928-cross-lingual-language-model-pretraining>.
- [72] Alexis Conneau, Guillaume Lample, Marc’Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. “Word translation without parallel data.” In: *arXiv preprint arXiv:1710.04087* (2017).
- [73] Yaim Cooper. “The loss landscape of overparameterized neural networks.” In: *arXiv preprint arXiv:1804.10200* (2018).
- [74] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. “Training deep neural networks with low precision multiplications.” In: *arXiv preprint arXiv:1412.7024* (2014).
- [75] Jia Cui, Brian Kingsbury, Bhuvana Ramabhadran, George Saon, Tom Sercu, Kartik Audhkhasi, Abhinav Sethy, Markus Nussbaum-Thom, and Andrew Rosenberg. “Knowledge distillation across ensembles of multilingual models for low-resource languages.” In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2017, pp. 4825–4829.
- [76] Yin Cui, Guandao Yang, Andreas Veit, Xun Huang, and Serge Belongie. “Learning to evaluate image captioning.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5804–5812.
- [77] Raj Dabre and Atsushi Fujita. “Recurrent stacking of layers for compact neural machine translation models.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 6292–6299.
- [78] Bin Dai, Chen Zhu, and David Wipf. “Compressing neural networks using the variational information bottleneck.” In: *arXiv preprint arXiv:1802.10399* (2018).
- [79] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. “NeST: A neural network synthesis tool based on a grow-and-prune paradigm.” In: *IEEE Transactions on Computers* 68.10 (2019), pp. 1487–1497.
- [80] Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. “Transformer-xl: Attentive language models beyond a fixed-length context.” In: *arXiv preprint arXiv:1901.02860* (2019).
- [81] William Dally. “High-performance hardware for machine learning.” In: *NIPS Tutorial* (2015).

- [82] Dipankar Das, Naveen Mellempudi, Dheevatsa Mudigere, Dhiraj Kalamkar, Sasikanth Avancha, Kunal Banerjee, Srinivas Sridharan, Karthik Vaidyanathan, Bharat Kaul, Evangelos Georganas, et al. “Mixed precision training of convolutional neural networks using integer operations.” In: *arXiv preprint arXiv:1802.00930* (2018).
- [83] Lieven De Lathauwer. “Decompositions of a higher-order tensor in block terms—Part II: Definitions and uniqueness.” In: *SIAM Journal on Matrix Analysis and Applications* 30.3 (2008), pp. 1033–1066.
- [84] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. “Universal transformers.” In: *arXiv preprint arXiv:1807.03819* (2018).
- [85] Michael Denkowski and Alon Lavie. “Meteor universal: Language specific translation evaluation for any target language.” In: *Proceedings of the ninth workshop on statistical machine translation*. 2014, pp. 376–380.
- [86] Michal Dereziński, Feynman Liang, and Michael W Mahoney. “Exact expressions for double descent and implicit regularization via surrogate random design.” In: *arXiv preprint arXiv:1912.04533* (2019).
- [87] Thomas Desautels, Andreas Krause, and Joel W Burdick. “Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization.” In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 3873–3923.
- [88] Tim Dettmers. “8-bit approximations for parallelism in deep learning.” In: *arXiv preprint arXiv:1511.04561* (2015).
- [89] Tim Dettmers and Luke Zettlemoyer. “Sparse networks from scratch: Faster training without losing performance.” In: *arXiv preprint arXiv:1907.04840* (2019).
- [90] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding.” In: *arXiv preprint arXiv:1810.04805* (2018).
- [91] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. 2019, pp. 4171–4186. DOI: [10.18653/v1/n19-1423](https://doi.org/10.18653/v1/n19-1423). URL: <https://doi.org/10.18653/v1/n19-1423>.
- [92] Paramveer S Dhillon, Dean P Foster, and Lyle H Ungar. “Eigenwords: spectral word embeddings.” In: *Journal of Machine Learning Research* 16 (2015), pp. 3035–3078.

- [93] Xiaohan Ding, Guiguang Ding, Jungong Han, and Sheng Tang. “Auto-balanced filter pruning for efficient convolutional neural networks.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 2018.
- [94] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. “Long-term recurrent convolutional networks for visual recognition and description.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 2625–2634.
- [95] Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. “Multi-task learning for multiple language translation.” In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Vol. 1. 2015, pp. 1723–1732.
- [96] Xin Dong, Shangyu Chen, and Sinno Pan. “Learning to prune deep neural networks via layer-wise optimal brain surgeon.” In: *Advances in Neural Information Processing Systems*. 2017, pp. 4857–4867.
- [97] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. “Hawq: Hessian aware quantization of neural networks with mixed-precision.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 293–302.
- [98] Yerai Doval, Jose Camacho-Collados, Luis Espinosa-Anke, and Steven Schockaert. “Improving cross-lingual word embeddings by meeting in the middle.” In: *arXiv preprint arXiv:1808.08780* (2018).
- [99] Simon S Du, Xiyu Zhai, Barnabas Póczos, and Aarti Singh. “Gradient descent provably optimizes over-parameterized neural networks.” In: *arXiv preprint arXiv:1810.02054* (2018).
- [100] David Eigen, Jason Rolfe, Rob Fergus, and Yann LeCun. “Understanding deep architectures using a recursive convolutional network.” In: *arXiv preprint arXiv:1312.1847* (2013).
- [101] Julian Eisenschlos, Sebastian Ruder, Piotr Czapla, Marcin Karadas, Sylvain Gugger, and Jeremy Howard. “MultiFiT: Efficient Multi-lingual Language Model Fine-tuning.” In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. 2019, pp. 5701–5706. DOI: [10.18653/v1/D19-1572](https://doi.org/10.18653/v1/D19-1572). URL: <https://doi.org/10.18653/v1/D19-1572>.

- [102] Andries Petrus Engelbrecht. “A new pruning heuristic based on variance analysis of sensitivity information.” In: *IEEE transactions on Neural Networks* 12.6 (2001), pp. 1386–1399.
- [103] Dafydd Evans. “A computationally efficient estimator for mutual information.” In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 464.2093 (2008), pp. 1203–1215.
- [104] Angela Fan, Edouard Grave, and Armand Joulin. “Reducing transformer depth on demand with structured dropout.” In: *arXiv preprint arXiv:1909.11556* (2019).
- [105] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Remi Gribonval, Herve Jegou, and Armand Joulin. *Training with Quantization Noise for Extreme Model Compression*. 2020. arXiv: [2004.07320](https://arxiv.org/abs/2004.07320) [cs.LG].
- [106] Hongchao Fang and Pengtao Xie. “CERT: Contrastive Self-supervised Learning for Language Understanding.” In: *arXiv preprint arXiv:2005.12766* (2020).
- [107] Yuwei Fang, Shuohang Wang, Zhe Gan, Siqi Sun, and Jingjing Liu. “FILTER: An enhanced fusion method for cross-lingual language understanding.” In: *arXiv preprint arXiv:2009.05166* (2020).
- [108] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppín. “Placing search in context: The concept revisited.” In: *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 406–414.
- [109] Philippe Flajolet, Daniele Gardy, and Loÿs Thimonier. “Birthday paradox, coupon collectors, caching algorithms and self-organizing search.” In: *Discrete Applied Mathematics* 39.3 (1992), pp. 207–229.
- [110] Peter J Forrester and Mario Kieburg. “Relating the Bures measure to the Cauchy two-matrix model.” In: *Communications in Mathematical Physics* 342.1 (2016), pp. 151–187.
- [111] Jonathan Frankle and Michael Carbin. “The lottery ticket hypothesis: Finding sparse, trainable neural networks.” In: *arXiv preprint arXiv:1803.03635* (2018).
- [112] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. “The lottery ticket hypothesis at scale.” In: *arXiv preprint arXiv:1903.01611* 8 (2019).
- [113] Markus Freitag, Yaser Al-Onaizan, and Baskaran Sankaran. “Ensemble distillation for neural machine translation.” In: *arXiv preprint arXiv:1702.01802* (2017).

- [114] Andrea Frome, Greg Corrado, Jonathon Shlens, Samy Bengio, Jeffrey Dean, Marc'Aurelio Ranzato, and Tomas Mikolov. "DeViSE: A Deep Visual-Semantic Embedding Model." In: *Neural Information Processing Systems (NIPS)*. 2013.
- [115] Kunihiro Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." In: *Biological cybernetics* 36.4 (1980), pp. 193–202.
- [116] Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. "Born again neural networks." In: *International Conference on Machine Learning*. PMLR. 2018, pp. 1607–1616.
- [117] Adam Gaier and David Ha. "Weight agnostic neural networks." In: *Advances in Neural Information Processing Systems*. 2019, pp. 5364–5378.
- [118] Yarín Gal and Zoubin Ghahramani. "A theoretically grounded application of dropout in recurrent neural networks." In: *Advances in neural information processing systems*. 2016, pp. 1019–1027.
- [119] Bin Gao, Jiang Bian, and Tie-Yan Liu. "Wordrep: A benchmark for research on learning word representations." In: *arXiv preprint arXiv:1407.1640* (2014).
- [120] Iker García, Rodrigo Agerri, and German Rigau. "A Common Semantic Space for Monolingual and Cross-Lingual Meta-Embeddings." In: *arXiv preprint arXiv:2001.06381* (2020).
- [121] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. "Convolutional sequence to sequence learning." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1243–1252.
- [122] Dedre Gentner and Kenneth D Forbus. "Computational models of analogy." In: *Wiley interdisciplinary reviews: cognitive science* 2.3 (2011), pp. 266–276.
- [123] Dedre Gentner, Keith J Holyoak, Keith James Holyoak, and Boicho N Kokinov. *The analogical mind: Perspectives from cognitive science*. MIT press, 2001.
- [124] Josu Goikoetxea, Eneko Agirre, and Aitor Soroa. "Single or Multiple? Combining Word Representations Independently Learned from Text and WordNet." In: *Proc. of AAAI*. 2016, pp. 2608–2614.
- [125] David E Goldberg and Kalyanmoy Deb. "A comparative analysis of selection schemes used in genetic algorithms." In: *Foundations of genetic algorithms*. Vol. 1. Elsevier, 1991, pp. 69–93.

- [126] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. “Differentiable soft quantization: Bridging full-precision and low-bit neural networks.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 4852–4861.
- [127] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. “Multi-digit number recognition from street view imagery using deep convolutional neural networks.” In: *arXiv preprint arXiv:1312.6082* (2013).
- [128] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative adversarial nets.” In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.
- [129] Mitchell A Gordon, Kevin Duh, and Nicholas Andrews. “Compressing bert: Studying the effects of weight pruning on transfer learning.” In: *arXiv preprint arXiv:2002.08307* (2020).
- [130] Kartik Goyal, Chris Dyer, and Taylor Berg-Kirkpatrick. “Differentiable scheduled sampling for credit assignment.” In: *arXiv preprint arXiv:1704.06970* (2017).
- [131] Yves Grandvalet and Yoshua Bengio. “Semi-supervised learning by entropy minimization.” In: *Advances in neural information processing systems*. 2005, pp. 529–536.
- [132] Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. “Efficient softmax approximation for GPUs.” In: *arXiv preprint arXiv:1609.04309* (2016).
- [133] Erica Greene. “Extracting structured data from recipes using conditional random fields.” In: *The New York Times Open Blog* (2015).
- [134] Qiushan Guo, Zhipeng Yu, Yichao Wu, Ding Liang, Haoyu Qin, and Junjie Yan. “Dynamic recursive neural network.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5147–5156.
- [135] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. “Deep learning with limited numerical precision.” In: *International Conference on Machine Learning*. 2015, pp. 1737–1746.
- [136] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. “Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks.” In: *IEEE transactions on neural networks and learning systems* 29.11 (2018), pp. 5784–5789.

- [137] Masafumi Hagiwara. “Removal of hidden units and weights for back propagation networks.” In: *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*. Vol. 1. IEEE. 1993, pp. 351–354.
- [138] Guy Halawi, Gideon Dror, Evgeniy Gabrilovich, and Yehuda Koren. “Large-scale learning of word relatedness with constraints.” In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 1406–1414.
- [139] Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions.” In: *SIAM review* 53.2 (2011), pp. 217–288.
- [140] Richard W Hamming. “Error detecting and error correcting codes.” In: *Bell System technical journal* 29.2 (1950), pp. 147–160.
- [141] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. “Co-teaching: Robust training of deep neural networks with extremely noisy labels.” In: *Advances in neural information processing systems*. 2018, pp. 8527–8537.
- [142] Hong-Gui Han and Jun-Fei Qiao. “A structure optimisation algorithm for feedforward neural network construction.” In: *Neurocomputing* 99 (2013), pp. 347–357.
- [143] S Han, H Mao, and WJ Dally. “Compressing deep neural networks with pruning, trained quantization and huffman coding.” In: *arXiv preprint* (2015).
- [144] Song Han, Huizi Mao, and William J Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding.” In: *arXiv preprint arXiv:1510.00149* (2015).
- [145] Song Han, Jeff Pool, John Tran, and William Dally. “Learning both weights and connections for efficient neural network.” In: *Advances in neural information processing systems*. 2015, pp. 1135–1143.
- [146] Ben Harwood, Vijay Kumar BG, Gustavo Carneiro, Ian Reid, and Tom Drummond. “Smart mining for deep metric learning.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2821–2829.
- [147] Babak Hassibi and David G Stork. “Second order derivatives for network pruning: Optimal brain surgeon.” In: *Advances in neural information processing systems*. 1993, pp. 164–171.

- [148] Babak Hassibi, David G Stork, and Gregory Wolff. “Optimal brain surgeon: Extensions and performance comparisons.” In: *Advances in neural information processing systems*. 1994, pp. 263–270.
- [149] Jingyi He, KC Tsolis, Kian Kenyon-Dean, and Jackie Chi Kit Cheung. “Learning Efficient Task-Specific Meta-Embeddings with Word Prisms.” In: *arXiv preprint arXiv:2011.02944* (2020).
- [150] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. “Momentum contrast for unsupervised visual representation learning.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9729–9738.
- [151] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep residual learning for image recognition.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [152] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Identity mappings in deep residual networks.” In: *European conference on computer vision*. Springer. 2016, pp. 630–645.
- [153] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. “Amc: Automl for model compression and acceleration on mobile devices.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 784–800.
- [154] Srinidhi Hegde, Ranjitha Prasad, Ramya Hebbalaguppe, and Vishwajith Kumar. “Variational Student: Learning Compact and Sparser Networks in Knowledge Distillation Framework.” In: *arXiv preprint arXiv:1910.12061* (2019).
- [155] Ernst Hellinger. “Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen.” In: *Journal für die reine und angewandte Mathematik (Crelles Journal)* 1909.136 (1909), pp. 210–271.
- [156] Olivier Henaff. “Data-efficient image recognition with contrastive predictive coding.” In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4182–4192.
- [157] Byeongho Heo, Minsik Lee, Sangdoon Yun, and Jin Young Choi. “Knowledge transfer via distillation of activation boundaries formed by hidden neurons.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, pp. 3779–3787.
- [158] Felix Hill, Roi Reichart, and Anna Korhonen. “Simlex-999: Evaluating semantic models with (genuine) similarity estimation.” In: *Computational Linguistics* 41.4 (2015), pp. 665–695.
- [159] Felix Hill, Roi Reichart, and Anna Korhonen. “Simlex-999: Evaluating semantic models with (genuine) similarity estimation.” In: *Computational Linguistics* (2016).

- [160] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. “Improving neural networks by preventing co-adaptation of feature detectors.” In: *arXiv preprint arXiv:1207.0580* (2012).
- [161] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. “Neural networks for machine learning.” In: *Coursera, video lectures 264* (2012), p. 1.
- [162] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. “Distilling the knowledge in a neural network.” In: *arXiv preprint arXiv:1503.02531* (2015).
- [163] Frank L Hitchcock. “The expression of a tensor or a polyadic as a sum of products.” In: *Journal of Mathematics and Physics* 6.1-4 (1927), pp. 164–189.
- [164] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. “Learning deep representations by mutual information estimation and maximization.” In: *arXiv preprint arXiv:1808.06670* (2018).
- [165] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory.” In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [166] Lu Hou, Quanming Yao, and James T Kwok. “Loss-aware binarization of deep networks.” In: *arXiv preprint arXiv:1611.01600* (2016).
- [167] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzkebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. “Parameter-Efficient Transfer Learning for NLP.” In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019, pp. 2790–2799. URL: <http://proceedings.mlr.press/v97/houlsby19a.html>.
- [168] Jeremy Howard and Sebastian Ruder. “Universal Language Model Fine-tuning for Text Classification.” In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018*. 2018, pp. 328–339. DOI: [10.18653/v1/P18-1031](https://doi.org/10.18653/v1/P18-1031). URL: <https://www.aclweb.org/anthology/P18-1031/>.
- [169] Jeremy Howard and Sebastian Ruder. “Universal language model fine-tuning for text classification.” In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2018, pp. 328–339.
- [170] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. “Convolutional neural network architectures for matching natural language sentences.” In: *Advances in neural information processing systems*. 2014, pp. 2042–2050.

- [171] Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. “XTREME: A Massively Multilingual Multi-task Benchmark for Evaluating Cross-lingual Generalization.” In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 12-18 July 2020, Virtual Conference*. 2020. arXiv: 2003.11080. URL: <https://arxiv.org/abs/2003.11080>.
- [172] Yiming Hu, Siyang Sun, Jianquan Li, Xingang Wang, and Qingyi Gu. “A novel channel pruning method for deep neural network compression.” In: *arXiv preprint arXiv:1805.11394* (2018).
- [173] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. “Improving word representations via global context and multiple word prototypes.” In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2012, pp. 873–882.
- [174] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. “Densely connected convolutional networks.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.
- [175] Haoyang Huang, Yaobo Liang, Nan Duan, Ming Gong, Linjun Shou, Daxin Jiang, and Ming Zhou. “Unicoder: A universal language encoder by pre-training with multiple cross-lingual tasks.” In: *Empirical Methods in Natural Language Processing*. 2019.
- [176] Zehao Huang and Naiyan Wang. “Like what you like: Knowledge distill via neuron selectivity transfer.” In: *arXiv preprint arXiv:1707.01219* (2017).
- [177] Ferenc Huszár. “How (not) to train your generative model: Scheduled sampling, likelihood, adversary?” In: *arXiv preprint arXiv:1511.05101* (2015).
- [178] Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. “Densenet: Implementing efficient convnet descriptor pyramids.” In: *arXiv preprint arXiv:1404.1869* (2014).
- [179] Hakan Inan, Khashayar Khosravi, and Richard Socher. “Tying word vectors and word classifiers: A loss framework for language modeling.” In: *arXiv preprint arXiv:1611.01462* (2016).
- [180] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift.” In: *arXiv preprint arXiv:1502.03167* (2015).

- [181] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. “Quantization and training of neural networks for efficient integer-arithmetic-only inference.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 2704–2713.
- [182] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. “Speeding up convolutional neural networks with low rank expansions.” In: *arXiv preprint arXiv:1405.3866* (2014).
- [183] Eric Jang, Shixiang Gu, and Ben Poole. “Categorical reparameterization with gumbel-softmax.” In: *arXiv preprint arXiv:1611.01144* (2016).
- [184] Pratik Jawanpuria, NTV Dev, Anoop Kunchukuttan, and Bamdev Mishra. “Learning geometric word meta-embeddings.” In: *arXiv preprint arXiv:2004.09219* (2020).
- [185] Herve Jegou, Matthijs Douze, and Cordelia Schmid. “Product quantization for nearest neighbor search.” In: *IEEE transactions on pattern analysis and machine intelligence* 33.1 (2010), pp. 117–128.
- [186] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. “Tinybert: Distilling bert for natural language understanding.” In: *arXiv preprint arXiv:1909.10351* (2019).
- [187] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. “Bag of tricks for efficient text classification.” In: *arXiv preprint arXiv:1607.01759* (2016).
- [188] David A Jurgens, Peter D Turney, Saif M Mohammad, and Keith J Holyoak. “Semeval-2012 task 2: Measuring degrees of relational similarity.” In: *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*. Association for Computational Linguistics. 2012, pp. 356–364.
- [189] Thomas Kailath. *Linear systems*. Vol. 156. Prentice-Hall Englewood Cliffs, NJ, 1980.
- [190] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharm Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. “A study of bfloat16 for deep learning training.” In: *arXiv preprint arXiv:1905.12322* (2019).
- [191] Ehud D Karnin. “A simple procedure for pruning back-propagation trained neural networks.” In: *IEEE transactions on neural networks* 1.2 (1990), pp. 239–242.

- [192] Andrej Karpathy and Li Fei-Fei. “Deep visual-semantic alignments for generating image descriptions.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3128–3137.
- [193] James Kennedy and Russell Eberhart. “Particle swarm optimization.” In: *Proceedings of ICNN’95-International Conference on Neural Networks*. Vol. 4. IEEE. 1995, pp. 1942–1948.
- [194] Tom Kenter and Maarten De Rijke. “Short text similarity with word embeddings.” In: *Proceedings of the 24th ACM international on conference on information and knowledge management*. ACM. 2015, pp. 1411–1420.
- [195] Ashraf M Kibriya and Eibe Frank. “An empirical comparison of exact nearest neighbour algorithms.” In: *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer. 2007, pp. 140–151.
- [196] Douwe Kiela, Changhan Wang, and Kyunghyun Cho. “Dynamic Meta-Embeddings for Improved Sentence Representations.” In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 1466–1477. URL: <https://www.aclweb.org/anthology/D18-1176.pdf>.
- [197] Mert Kilickaya, Aykut Erdem, Nazli Ikizler-Cinbis, and Erkut Erdem. “Re-evaluating automatic metrics for image captioning.” In: *arXiv preprint arXiv:1612.07600* (2016).
- [198] Jangho Kim, SeongUk Park, and Nojun Kwak. “Paraphrasing complex network: Network compression via factor transfer.” In: *Advances in neural information processing systems*. 2018, pp. 2760–2769.
- [199] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. “Deeply-recursive convolutional network for image super-resolution.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1637–1645.
- [200] Yoon Kim. “Convolutional neural networks for sentence classification.” In: *arXiv preprint arXiv:1408.5882* (2014).
- [201] Yoon Kim and Alexander M Rush. “Sequence-level knowledge distillation.” In: *arXiv preprint arXiv:1606.07947* (2016).
- [202] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization.” In: *arXiv preprint arXiv:1412.6980* (2014).
- [203] Diederik P Kingma, Tim Salimans, and Max Welling. “Variational dropout and the local reparameterization trick.” In: *Advances in Neural Information Processing Systems*. 2015, pp. 2575–2583.

- [204] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. “Skip-thought vectors.” In: *Advances in neural information processing systems*. 2015, pp. 3294–3302.
- [205] Okan Köpüklü, Maryam Babae, Stefan Hörmann, and Gerhard Rigoll. “Convolutional neural networks with layer reuse.” In: *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2019, pp. 345–349.
- [206] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. “Similarity of neural network representations revisited.” In: *arXiv preprint arXiv:1905.00414* (2019).
- [207] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. “Estimating mutual information.” In: *Physical review E* 69.6 (2004), p. 066138.
- [208] Raghuraman Krishnamoorthi. “Quantizing deep convolutional networks for efficient inference: A whitepaper.” In: *arXiv preprint arXiv:1806.08342* (2018).
- [209] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images.” In: *cs.toronto.edu* (2009).
- [210] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks.” In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.
- [211] Taku Kudo and John Richardson. “Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing.” In: *arXiv preprint arXiv:1808.06226* (2018).
- [212] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. “From word embeddings to document distances.” In: *International Conference on Machine Learning*. 2015, pp. 957–966.
- [213] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. “Race: Large-scale reading comprehension dataset from examinations.” In: *arXiv preprint arXiv:1704.04683* (2017).
- [214] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. “Recurrent Convolutional Neural Networks for Text Classification.” In: *AAAI*. Vol. 333. 2015, pp. 2267–2273.
- [215] Samuli Laine and Timo Aila. “Temporal ensembling for semi-supervised learning.” In: *arXiv preprint arXiv:1610.02242* (2016).
- [216] Guillaume Lample and Alexis Conneau. “Cross-lingual language model pretraining.” In: *arXiv preprint arXiv:1901.07291* (2019).

- [217] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. “Albert: A lite bert for self-supervised learning of language representations.” In: *arXiv preprint arXiv:1909.11942* (2019).
- [218] Philippe Lauret, Eric Fock, and Thierry Alex Mara. “A node pruning algorithm based on a Fourier amplitude sensitivity test method.” In: *IEEE transactions on neural networks* 17.2 (2006), pp. 273–293.
- [219] Jimmie D Lawson and Yongdo Lim. “The geometric mean, matrices, metrics, and more.” In: *The American Mathematical Monthly* 108.9 (2001), pp. 797–812.
- [220] Yann LeCun. “The MNIST database of handwritten digits.” In: <http://yann.lecun.com/exdb/mnist/> (1998).
- [221] Yann LeCun, Yoshua Bengio, et al. “Convolutional networks for images, speech, and time series.” In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.
- [222] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition.” In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [223] Yann LeCun, John S Denker, and Sara A Solla. “Optimal brain damage.” In: *Advances in neural information processing systems*. 1990, pp. 598–605.
- [224] Vadim Lebedev and Victor Lempitsky. “Fast convnets using group-wise brain damage.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2554–2564.
- [225] Rémi Lebreton and Ronan Collobert. “Word embeddings through hellinger PCA.” In: *arXiv preprint arXiv:1312.5542* (2013).
- [226] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. “Layer-adaptive Sparsity for the Magnitude-based Pruning.” In: *International Conference on Learning Representations*. 2020.
- [227] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. “Snip: Single-shot network pruning based on connection sensitivity.” In: *arXiv preprint arXiv:1810.02340* (2018).
- [228] Erich Leo Lehmann. *Elements of large-sample theory*. Springer Science & Business Media, 2004.
- [229] Chee Wee Leong and Rada Mihalcea. “Measuring the semantic relatedness between words and images.” In: *Proceedings of the Ninth International Conference on Computational Semantics (IWCS 2011)*. 2011.

- [230] Asriel U Levin, Todd K Leen, and John E Moody. “Fast pruning using principal components.” In: *Advances in neural information processing systems*. 1994, pp. 35–42.
- [231] Omer Levy, Yoav Goldberg, and Ido Dagan. “Improving Distributional Similarity with Lessons Learned from Word Embeddings.” In: *Transactions of the Association for Computational Linguistics* 3 (2015), pp. 211–225.
- [232] Patrick Lewis, Barlas Oğuz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. “MLQA: Evaluating cross-lingual extractive question answering.” In: *Association for Computational Linguistics*. 2020.
- [233] Fengfu Li, Bo Zhang, and Bin Liu. “Ternary weight networks.” In: *arXiv preprint arXiv:1605.04711* (2016).
- [234] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. “Pruning filters for efficient convnets.” In: *arXiv preprint arXiv:1608.08710* (2016).
- [235] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. “Visualizing the loss landscape of neural nets.” In: *Advances in Neural Information Processing Systems*. 2018, pp. 6389–6399.
- [236] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E Hopcroft. “Convergent Learning: Do different neural networks learn the same representations?” In: *Iclr*. 2016.
- [237] Yuncheng Li, Jianchao Yang, Yale Song, Liangliang Cao, Jiebo Luo, and Li-Jia Li. “Learning from noisy labels with distillation.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1910–1918.
- [238] Yaobo Liang, Nan Duan, Yeyun Gong, Ning Wu, Fenfei Guo, Weizhen Qi, Ming Gong, Linjun Shou, Daxin Jiang, Guihong Cao, et al. “XGLUE: A new benchmark dataset for cross-lingual pre-training, understanding and generation.” In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2020, pp. 6008–6018.
- [239] Chin-Yew Lin. “Rouge: A package for automatic evaluation of summaries.” In: *Text Summarization Branches Out* (2004).
- [240] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. “Fixed point quantization of deep convolutional networks.” In: *International Conference on Machine Learning*. 2016, pp. 2849–2858.
- [241] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. “Runtime neural pruning.” In: *Advances in Neural Information Processing Systems*. 2017, pp. 2181–2191.

- [242] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. “Accelerating Convolutional Networks via Global & Dynamic Filter Pruning.” In: *IJCAI*. 2018, pp. 2425–2432.
- [243] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. “Towards optimal structured cnn pruning via generative adversarial learning.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2790–2799.
- [244] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. “Microsoft coco: Common objects in context.” In: *European conference on computer vision*. Springer. 2014, pp. 740–755.
- [245] Xiaofan Lin, Cong Zhao, and Wei Pan. “Towards accurate binary convolutional neural network.” In: *Advances in Neural Information Processing Systems*. 2017, pp. 345–353.
- [246] Ralph Linsker. “An application of the principle of maximum information preservation to linear systems.” In: *Advances in neural information processing systems* 1 (1988), pp. 186–194.
- [247] Chenxi Liu, Junhua Mao, Fei Sha, and Alan Yuille. “Attention correctness in neural image captioning.” In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.
- [248] Chia-Wei Liu, Ryan Lowe, Iulian V Serban, Michael Noseworthy, Laurent Charlin, and Joelle Pineau. “How not to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation.” In: *arXiv preprint arXiv:1603.08023* (2016).
- [249] Hanxiao Liu, Karen Simonyan, and Yiming Yang. “Darts: Differentiable architecture search.” In: *arXiv preprint arXiv:1806.09055* (2018).
- [250] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. “Understanding the difficulty of training transformers.” In: *arXiv preprint arXiv:2004.08249* (2020).
- [251] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. “Recurrent neural network for text classification with multi-task learning.” In: *arXiv preprint arXiv:1605.05101* (2016).
- [252] Siqi Liu, Zhenhai Zhu, Ning Ye, Sergio Guadarrama, and Kevin Murphy. “Optimization of image description metrics using policy gradient methods.” In: *CoRR* abs/1612.00370 (2016). arXiv: 1612.00370. URL: <http://arxiv.org/abs/1612.00370>.
- [253] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-Yi Wang. “Representation Learning Using Multi-Task Deep Neural Networks for Semantic Classification and Information Retrieval.” In: *NAACL*. 2015.

- [254] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. “Improving multi-task deep neural networks via knowledge distillation for natural language understanding.” In: *arXiv preprint arXiv:1904.09482* (2019).
- [255] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. “Roberta: A robustly optimized bert pretraining approach.” In: *arXiv preprint arXiv:1907.11692* (2019).
- [256] Yuhua Liu, Saurabh Agarwal, and Shivaram Venkataraman. “AutoFreeze: Automatically Freezing Model Blocks to Accelerate Fine-tuning.” In: *arXiv preprint arXiv:2102.01386* (2021).
- [257] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. “Learning efficient convolutional networks through network slimming.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2736–2744.
- [258] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. “Rethinking the value of network pruning.” In: *arXiv preprint arXiv:1810.05270* (2018).
- [259] Stuart Lloyd. “Least squares quantization in PCM.” In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.
- [260] Lajanugen Logeswaran and Honglak Lee. “An efficient framework for learning sentence representations.” In: *arXiv preprint arXiv:1803.02893* (2018).
- [261] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.
- [262] Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. “Data-free knowledge distillation for deep neural networks.” In: *arXiv preprint arXiv:1710.07535* (2017).
- [263] Christos Louizos, Karen Ullrich, and Max Welling. “Bayesian compression for deep learning.” In: *Advances in Neural Information Processing Systems*. 2017, pp. 3288–3298.
- [264] Christos Louizos, Max Welling, and Diederik P Kingma. “Learning Sparse Neural Networks through  $L_0$  Regularization.” In: *arXiv preprint arXiv:1712.01312* (2017).
- [265] Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. “Addressing the rare word problem in neural machine translation.” In: *arXiv preprint arXiv:1410.8206* (2014).

- [266] Thang Luong, Richard Socher, and Christopher D Manning. “Better word representations with recursive neural networks for morphology.” In: *CoNLL*. 2013, pp. 104–113.
- [267] Xuezhe Ma, Pengcheng Yin, Jingzhou Liu, Graham Neubig, and Eduard Hovy. “Softmax q-distribution estimation for structured prediction: A theoretical interpretation for raml.” In: *arXiv preprint arXiv:1705.07136* (2017).
- [268] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. “Learning word vectors for sentiment analysis.” In: *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*. Association for Computational Linguistics. 2011, pp. 142–150.
- [269] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. “The concrete distribution: A continuous relaxation of discrete random variables.” In: *arXiv preprint arXiv:1611.00712* (2016).
- [270] Arun Mallya and Svetlana Lazebnik. “Piggyback: Adding multiple tasks to a single, fixed network by learning to mask.” In: *arXiv preprint arXiv:1801.06519* 6.8 (2018).
- [271] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. “Deep captioning with multimodal recurrent neural networks (m-rnn).” In: *arXiv preprint arXiv:1412.6632* (2014).
- [272] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. “Building a large annotated corpus of English: The Penn Treebank.” In: *UPenn Repository* (1993).
- [273] William H Marlow et al. “Factorial distributions.” In: *The Annals of Mathematical Statistics* 36.3 (1965), pp. 1066–1068.
- [274] James Martens and Roger Grosse. “Optimizing neural networks with kronecker-factored approximate curvature.” In: *International conference on machine learning*. PMLR. 2015, pp. 2408–2417.
- [275] Gábor Melis, Chris Dyer, and Phil Blunsom. “On the state of the art of evaluation in neural language models.” In: *arXiv preprint arXiv:1707.05589* (2017).
- [276] Naveen Mellempudi, Sudarshan Srinivasan, Dipankar Das, and Bharat Kaul. “Mixed precision training with 8-bit floating point.” In: *arXiv preprint arXiv:1905.12334* (2019).
- [277] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. “Pointer sentinel mixture models.” In: *arXiv preprint arXiv:1609.07843* (2016).

- [278] Paul Merolla, Rathinakumar Appuswamy, John Arthur, Steve K Esser, and Dharmendra Modha. “Deep neural networks are robust to weight binarization and other non-linear distortions.” In: *arXiv preprint arXiv:1606.01981* (2016).
- [279] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. “Mixed precision training.” In: *arXiv preprint arXiv:1710.03740* (2017).
- [280] Szymon Migacz. “8-bit inference with tensorrt.” In: *GPU technology conference*. Vol. 2. 2017, p. 5.
- [281] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space.” In: *arXiv preprint arXiv:1301.3781* (2013).
- [282] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. “Recurrent neural network based language model.” In: *Eleventh Annual Conference of the International Speech Communication Association*. 2010.
- [283] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. “Distributed representations of words and phrases and their compositionality.” In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.
- [284] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. “Linguistic regularities in continuous space word representations.” In: *hlt-Naacl*. Vol. 13. 2013, pp. 746–751.
- [285] G. Miller and W. Charles. “Contextual correlates of semantic similarity.” In: *Language and Cognitive Processes* 6(1) (1998), pp. 1–28.
- [286] Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, and Hassan Ghasemzadeh. “Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher.” In: *arXiv preprint arXiv:1902.03393* (2019).
- [287] Asit Mishra and Debbie Marr. “Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy.” In: *arXiv preprint arXiv:1711.05852* (2017).
- [288] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. “WRPN: wide reduced-precision networks.” In: *arXiv preprint arXiv:1709.01134* (2017).
- [289] Andriy Mnih and Geoffrey E Hinton. “A scalable hierarchical distributed language model.” In: *Advances in neural information processing systems* 21 (2008), pp. 1081–1088.
- [290] Andriy Mnih and Yee Whye Teh. “A fast and simple algorithm for training neural probabilistic language models.” In: *arXiv preprint arXiv:1206.6426* (2012).

- [291] Maher Moakher and Philipp G Batchelor. “Symmetric positive-definite matrices: From geometry to applications and visualization.” In: *Visualization and Processing of Tensor Fields*. Springer, 2006, pp. 285–298.
- [292] Hossein Mobahi, Mehrdad Farajtabar, and Peter L Bartlett. “Self-distillation amplifies regularization in hilbert space.” In: *arXiv preprint arXiv:2002.05715* (2020).
- [293] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. “Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science.” In: *Nature communications* 9.1 (2018), pp. 1–12.
- [294] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. “Variational dropout sparsifies deep neural networks.” In: *arXiv preprint arXiv:1701.05369* (2017).
- [295] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. “Pruning convolutional neural networks for resource efficient transfer learning.” In: *arXiv preprint arXiv:1611.06440* (2016).
- [296] Alexander Monakov, Anton Lokhmotov, and Arutyun Avetisyan. “Automatically tuning sparse matrix-vector multiplication for GPU architectures.” In: *International Conference on High-Performance Embedded Architectures and Compilers*. Springer, 2010, pp. 111–125.
- [297] Frederic Morin and Yoshua Bengio. “Hierarchical probabilistic neural network language model.” In: *Aistats*. Vol. 5. Citeseer, 2005, pp. 246–252.
- [298] Hesham Mostafa and Xin Wang. “Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization.” In: *arXiv preprint arXiv:1902.05967* (2019).
- [299] Michael C Mozer and Paul Smolensky. “Skeletonization: A technique for trimming the fat from a network via relevance assessment.” In: *Advances in neural information processing systems*. 1989, pp. 107–115.
- [300] Jiaqi Mu, Suma Bhat, and Pramod Viswanath. “All-but-the-top: Simple and effective postprocessing for word representations.” In: *arXiv preprint arXiv:1702.01417* (2017).
- [301] Jonas Mueller and Aditya Thyagarajan. “Siamese Recurrent Architectures for Learning Sentence Similarity.” In: *AAAI*. 2016, pp. 2786–2792.
- [302] Benjamin Muller, Yanai Elazar, Benoît Sagot, and Djamé Seddah. “First Align, then Predict: Understanding the Cross-Lingual Ability of Multilingual BERT.” In: *arXiv preprint arXiv:2101.11109* (2021).

- [303] Rafael Muller, Simon Kornblith, and Geoffrey E Hinton. “When does label smoothing help?” In: *Advances in Neural Information Processing Systems*. 2019, pp. 4696–4705.
- [304] Avo Muromägi, Kairit Sirts, and Sven Laur. “Linear ensembles of word embedding models.” In: *arXiv preprint arXiv:1704.01419* (2017).
- [305] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. “Deep double descent: Where bigger models and more data hurt.” In: *arXiv preprint arXiv:1912.02292* (2019).
- [306] Pramod L Narasimha, Walter H Delashmit, Michael T Manry, Jiang Li, and Francisco Maldonado. “An integrated growing-pruning method for feedforward network training.” In: *Neurocomputing* 71.13-15 (2008), pp. 2831–2847.
- [307] James O’ Neill and Danushka Bollegala. “Meta-embedding as auxiliary task regularization.” In: *arXiv preprint arXiv:1809.05886* (2018).
- [308] James O’ Neill and Danushka Bollegala. “Semantically-Conditioned Negative Samples for Efficient Contrastive Learning.” In: *arXiv preprint arXiv:2102.06603* (2021).
- [309] James O’ Neill, Sourav Dutta, and Haytham Assem. “Deep Neural Compression Via Concurrent Pruning and Self-Distillation.” In: *arXiv preprint arXiv:2109.15014* (2021).
- [310] James O’ Neill, Sourav Dutta, and Haytham Assem. “Aligned Weight Regularizers for Pruning Pretrained Neural Networks.” In: *arXiv preprint arXiv:2204.01385* (2022).
- [311] Behnam Neyshabur, Zhiyuan Li, Srinadh Bhojanapalli, Yann LeCun, and Nathan Srebro. “Towards understanding the role of over-parametrization in generalization of neural networks.” In: *arXiv preprint arXiv:1805.12076* (2018).
- [312] Joakim Nivre, Marie-Catherine De Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D Manning, Ryan T McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, et al. “Universal Dependencies v1: A Multilingual Treebank Collection.” In: *LREC*. 2016.
- [313] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajič, Christopher D Manning, Sampo Pyysalo, Sebastian Schuster, Francis Tyers, and Daniel Zeman. “Universal dependencies v2: An evergrowing multilingual treebank collection.” In: *arXiv preprint arXiv:2004.10643* (2020).

- [314] Mohammad Norouzi, Samy Bengio, Navdeep Jaitly, Mike Schuster, Yonghui Wu, Dale Schuurmans, et al. “Reward augmented maximum likelihood for neural structured prediction.” In: *Advances In Neural Information Processing Systems* 29 (2016), pp. 1723–1731.
- [315] Alexander Novikov, Dmitrii Podoprikin, Anton Osokin, and Dmitry P Vetrov. “Tensorizing neural networks.” In: *Advances in neural information processing systems*. 2015, pp. 442–450.
- [316] Steven J Nowlan and Geoffrey E Hinton. “Simplifying neural networks by soft weight-sharing.” In: *Neural computation* 4.4 (1992), pp. 473–493.
- [317] Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik-Manor. “Asap: Architecture search, anneal and prune.” In: *arXiv preprint arXiv:1904.04123* (2019).
- [318] James O’Neill, Greg V Steeg, and Aram Galstyan. “Layer-Wise Neural Network Compression via Layer Fusion.” In: *Asian Conference on Machine Learning*. PMLR. 2021, pp. 1381–1396.
- [319] Yusuke Oda, Philip Arthur, Graham Neubig, Koichiro Yoshino, and Satoshi Nakamura. “Neural machine translation via binary code prediction.” In: *arXiv preprint arXiv:1704.06918* (2017).
- [320] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. “Deep metric learning via lifted structured feature embedding.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4004–4012.
- [321] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. “Representation learning with contrastive predictive coding.” In: *arXiv preprint arXiv:1807.03748* (2018).
- [322] A Emin Orhan and Xaq Pitkow. “Skip connections eliminate singularities.” In: *arXiv preprint arXiv:1701.09175* (2017).
- [323] Ivan V Oseledets. “Tensor-train decomposition.” In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2295–2317.
- [324] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. “BLEU: a method for automatic evaluation of machine translation.” In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318.
- [325] Daniel S. Park, Yu Zhang, Ye Jia, Wei Han, Chung-Cheng Chiu, Bo Li, Yonghui Wu, and Quoc V. Le. *Improved Noisy Student Training for Automatic Speech Recognition*. 2020. arXiv: 2005.09629 [eess.AS].

- [326] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. “Value-aware quantization for training and inference of neural networks.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 580–595.
- [327] Sejun Park, Jaeho Lee, Sangwoo Mo, and Jinwoo Shin. “Lookahead: A far-sighted alternative of magnitude-based pruning.” In: *arXiv preprint arXiv:2002.04809* (2020).
- [328] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. “Relational knowledge distillation.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3967–3976.
- [329] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. “Image transformer.” In: *arXiv preprint arXiv:1802.05751* (2018).
- [330] Nikolaos Passalis and Anastasios Tefas. “Learning deep representations with probabilistic knowledge transfer.” In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 268–284.
- [331] Nikolaos Passalis, Maria Tzelepi, and Anastasios Tefas. “Probabilistic Knowledge Transfer for Lightweight Deep Representation Learning.” In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).
- [332] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluís-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. “Carbon emissions and large neural network training.” In: *arXiv preprint arXiv:2104.10350* (2021).
- [333] Baoyun Peng, Xiao Jin, Jiaheng Liu, Dongsheng Li, Yichao Wu, Yu Liu, Shunfeng Zhou, and Zhaoning Zhang. “Correlation congruence for knowledge distillation.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 5007–5016.
- [334] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. “A Riemannian framework for tensor computing.” In: *International Journal of computer vision* 66.1 (2006), pp. 41–66.
- [335] Jeffrey Pennington, Richard Socher, and Christopher Manning. “Glove: Global vectors for word representation.” In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.
- [336] Matthew E Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. “Semi-supervised sequence tagging with bidirectional language models.” In: *arXiv preprint arXiv:1705.00108* (2017).

- [337] Matthew E Peters, Sebastian Ruder, and Noah A Smith. “To tune or not to tune? adapting pretrained representations to diverse tasks.” In: *arXiv preprint arXiv:1903.05987* (2019).
- [338] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. “AdapterFusion: Non-destructive task composition for transfer learning.” In: *arXiv preprint arXiv:2005.00247* (2020).
- [339] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. “AdapterHub: A Framework for Adapting Transformers.” In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 2020, pp. 46–54.
- [340] Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. “MAD-X: An Adapter-based Framework for Multi-task Cross-lingual Transfer.” In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Virtual Conference*. 2020. URL: <https://arxiv.org/pdf/2005.00052.pdf>.
- [341] Mary Phuong and Christoph Lampert. “Towards understanding knowledge distillation.” In: *International Conference on Machine Learning*. 2019, pp. 5142–5151.
- [342] Steven T Piantadosi. “Zipf’s word frequency law in natural language: A critical review and future directions.” In: *Psychonomic bulletin & review* 21.5 (2014), pp. 1112–1130.
- [343] Bryan A Plummer, Nikoli Dryden, Julius Frost, Torsten Hoeffler, and Kate Saenko. “Shapeshifter Networks: Cross-layer Parameter Sharing for Scalable and Effective Deep Learning.” In: *arXiv preprint arXiv:2006.10598* (2020).
- [344] Nina Poerner, Ulli Waltinger, and Hinrich Schütze. “Sentence meta-embeddings for unsupervised semantic textual similarity.” In: *arXiv preprint arXiv:1911.03700* (2019).
- [345] Robi Polikar. “Ensemble learning.” In: *Ensemble machine learning*. Springer, 2012, pp. 1–34.
- [346] Antonio Polino, Razvan Pascanu, and Dan Alistarh. “Model compression via distillation and quantization.” In: *arXiv preprint arXiv:1802.05668* (2018).
- [347] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. “Learning transferable visual models from natural language supervision.” In: *arXiv preprint arXiv:2103.00020* (2021).

- [348] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks.” In: *arXiv preprint arXiv:1511.06434* (2015).
- [349] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. “Improving language understanding by generative pre-training.” In: URL <https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/languageunsupervised/language-understanding-paper.pdf> (2018).
- [350] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. “Squad: 100,000+ questions for machine comprehension of text.” In: *arXiv preprint arXiv:1606.05250* (2016).
- [351] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. “What’s Hidden in a Randomly Weighted Neural Network?” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 11893–11902.
- [352] Marc’Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. “Sequence level training with recurrent neural networks.” In: *arXiv preprint arXiv:1511.06732* (2015).
- [353] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. “Xnor-net: Imagenet classification using binary convolutional neural networks.” In: *European conference on computer vision*. Springer. 2016, pp. 525–542.
- [354] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. “On the convergence of adam and beyond.” In: *arXiv preprint arXiv:1904.09237* (2018).
- [355] Russell Reed. “Pruning algorithms-a survey.” In: *IEEE transactions on Neural Networks* 4.5 (1993), pp. 740–747.
- [356] Shuo Ren, Yu Wu, Shujie Liu, Ming Zhou, and Shuai Ma. “Explicit cross-lingual pre-training for unsupervised machine translation.” In: *arXiv preprint arXiv:1909.00180* (2019).
- [357] Zhou Ren, Xiaoyu Wang, Ning Zhang, Xutao Lv, and Li-Jia Li. “Deep reinforcement learning-based image captioning with embedding reward.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 290–298.
- [358] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. “Self-critical sequence training for image captioning.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7008–7024.
- [359] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. “Learning separable filters.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 2754–2761.

- [360] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. “Fitnets: Hints for thin deep nets.” In: *arXiv preprint arXiv:1412.6550* (2014).
- [361] Sheldon Ross. *A first course in probability*. Pearson, 2014.
- [362] Stéphane Ross and Drew Bagnell. “Efficient reductions for imitation learning.” In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 661–668.
- [363] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. “A reduction of imitation learning and structured prediction to no-regret online learning.” In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 627–635.
- [364] C Rosset. “Turing-NLG: A 17-billion-parameter language model by microsoft.” In: *Microsoft Blog* (2019).
- [365] Herbert Rubenstein and John B Goodenough. “Contextual correlates of synonymy.” In: *Communications of the ACM* 8.10 (1965), pp. 627–633.
- [366] Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. “AdapterDrop: On the Efficiency of Adapters in Transformers.” In: *arXiv preprint arXiv:2010.11918* (2020).
- [367] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
- [368] Tara N Sainath, Brian Kingsbury, Vikas Sindhvani, Ebru Arisoy, and Bhuvana Ramabhadran. “Low-rank matrix factorization for deep neural network training with high-dimensional output targets.” In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6655–6659.
- [369] Alexandre Salle, Marco Idiart, and Aline Villavicencio. “Matrix factorization using window sampling and negative sampling for improved word representations.” In: *arXiv preprint arXiv:1606.00819* (2016).
- [370] Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming, portable documents*. Addison-Wesley Professional, 2010.
- [371] Erik F Sang and Fien De Meulder. “Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition.” In: *arXiv preprint cs/0306050* (2003).

- [372] Victor Sanh. *Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT*. 2019. URL: <https://medium.com/huggingface/distilbert-8cf3380435b5> (visited on 08/28/2019).
- [373] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.” In: *arXiv preprint arXiv:1910.01108* (2019).
- [374] Victor Sanh, Thomas Wolf, and Alexander M Rush. “Movement Pruning: Adaptive Sparsity by Fine-Tuning.” In: *arXiv preprint arXiv:2005.07683* (2020).
- [375] Bharat Bhushan Sau and Vineeth N Balasubramanian. “Deep model compression: Distilling knowledge from noisy teachers.” In: *arXiv preprint arXiv:1610.09650* (2016).
- [376] Pedro Savarese and Michael Maire. “Learning implicitly recurrent CNNs through parameter sharing.” In: *arXiv preprint arXiv:1902.09701* (2019).
- [377] Andrew M Saxe, Yamini Bansal, Joel Dapello, Madhu Advani, Artemy Kolchinsky, Brendan D Tracey, and David D Cox. “On the information bottleneck theory of deep learning.” In: *Journal of Statistical Mechanics: Theory and Experiment* 2019.12 (2019), p. 124020.
- [378] Jurgen Schmidhuber. “Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook.” PhD thesis. Technische Universitat Munchen, 1987.
- [379] Florian Schroff, Dmitry Kalenichenko, and James Philbin. “Facenet: A unified embedding for face recognition and clustering.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.
- [380] Terrence J Sejnowski and Charles R Rosenberg. “Parallel networks that learn to pronounce English text.” In: *Complex systems* 1.1 (1987), pp. 145–168.
- [381] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Neural machine translation of rare words with subword units.” In: *arXiv preprint arXiv:1508.07909* (2015).
- [382] Rudy Setiono and Wee Kheng Leow. “Pruned neural networks for regression.” In: *Pacific Rim International Conference on Artificial Intelligence*. Springer. 2000, pp. 500–509.
- [383] Anish Shah, Eashan Kadam, Hena Shah, Sameer Shinde, and Sandip Shingade. “Deep residual networks with exponential linear unit.” In: *arXiv preprint arXiv:1604.04112* (2016).

- [384] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. “Q-bert: Hessian based ultra low precision quantization of bert.” In: *arXiv preprint arXiv:1909.05840* (2019).
- [385] Kaiyu Shi and Kai Yu. “Structured Word Embedding for Low Memory Neural Network Language Model.” In: *Proc. Interspeech 2018* (2018), pp. 1254–1258.
- [386] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. “Hash kernels for structured data.” In: *Journal of Machine Learning Research* 10.Nov (2009), pp. 2615–2637.
- [387] Xing Shi, Kevin Knight, and Deniz Yuret. “Why neural translations are the right length.” In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 2016, pp. 2278–2282.
- [388] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. “Megatron-lm: Training multi-billion parameter language models using gpu model parallelism.” In: *arXiv preprint arXiv:1909.08053* (2019).
- [389] Raphael Shu and Hideki Nakayama. “Compressing Word Embeddings via Deep Compositional Code Learning.” In: *arXiv preprint arXiv:1711.01068* (2017).
- [390] Ravid Shwartz-Ziv and Naftali Tishby. “Opening the black box of deep neural networks via information.” In: *arXiv preprint arXiv:1703.00810* (2017).
- [391] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. “Deterministic policy gradient algorithms.” In: *ICML*. 2014.
- [392] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition.” In: *arXiv preprint arXiv:1409.1556* (2014).
- [393] Sidak Pal Singh and Dan Alistarh. “WoodFisher: Efficient second-order approximations for model compression.” In: *arXiv preprint arXiv:2004.14340* (2020).
- [394] Sidak Pal Singh and Martin Jaggi. “Model Fusion via Optimal Transport.” In: *arXiv preprint arXiv:1910.05653* (2019).
- [395] Xuemeng Song, Fuli Feng, Xianjing Han, Xin Yang, Wei Liu, and Liqiang Nie. “Neural compatibility modeling with attentive knowledge distillation.” In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 2018, pp. 5–14.

- [396] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting.” In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.
- [397] Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A Alemi, and Andrew Gordon Wilson. “Does Knowledge Distillation Really Work?” In: *arXiv preprint arXiv:2106.05945* (2021).
- [398] Pierre Stock, Armand Joulin, Remi Gribonval, Benjamin Graham, and Herve Jegou. “And the bit goes down: Revisiting the quantization of neural networks.” In: *arXiv preprint arXiv:1907.05686* (2019).
- [399] Emma Strubell, Ananya Ganesh, and Andrew McCallum. “Energy and Policy Considerations for Deep Learning in NLP.” In: *arXiv preprint arXiv:1906.02243* (2019).
- [400] Yumin Suh, Bohyung Han, Wonsik Kim, and Kyoung Mu Lee. “Stochastic class-based hard example mining for deep metric learning.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 7251–7259.
- [401] Fei Sun, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. “Learning word representations by jointly modeling syntagmatic and paradigmatic relations.” In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Vol. 1. 2015, pp. 136–145.
- [402] Siqu Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. “Patient knowledge distillation for bert model compression.” In: *arXiv preprint arXiv:1908.09355* (2019).
- [403] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. “LSTM neural networks for language modeling.” In: *Thirteenth annual conference of the international speech communication association*. 2012.
- [404] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. “Sequence to sequence learning with neural networks.” In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.
- [405] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. “Policy gradient methods for reinforcement learning with function approximation.” In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.
- [406] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. “Inception-v4, inception-resnet and the impact of residual connections on learning.” In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

- [407] Ying Tai, Jian Yang, and Xiaoming Liu. “Image super-resolution via deep recursive residual network.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3147–3155.
- [408] Mingxing Tan and Quoc V. Le. *EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling*. 2019. URL: <https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html> (visited on 05/29/2019).
- [409] Mingxing Tan and Quoc V Le. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.” In: *arXiv preprint arXiv:1905.11946* (2019).
- [410] Hidenori Tanaka, Daniel Kunin, Daniel LK Yamins, and Surya Ganguli. “Pruning neural networks without any data by iteratively conserving synaptic flow.” In: *arXiv preprint arXiv:2006.05467* (2020).
- [411] Chongyang Tao, Lili Mou, Dongyan Zhao, and Rui Yan. “Rubber: An unsupervised method for automatic evaluation of open-domain dialog systems.” In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.
- [412] Antti Tarvainen and Harri Valpola. “Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results.” In: *Advances in neural information processing systems*. 2017, pp. 1195–1204.
- [413] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. “Faster gaze prediction with dense networks and fisher pruning.” In: *arXiv preprint arXiv:1801.05787* (2018).
- [414] Yonglong Tian, Dilip Krishnan, and Phillip Isola. “Contrastive representation distillation.” In: *arXiv preprint arXiv:1910.10699* (2019).
- [415] Julien Tissier, Christophe Gravier, and Amaury Habrard. “Dict2vec: Learning word embeddings using lexical dictionaries.” In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017, pp. 254–263.
- [416] Viet-Anh Tran, Romain Hennequin, Jimena Royo-Letelier, and Manuel Moussallam. “Improving Collaborative Metric Learning with Efficient Negative Sampling.” In: *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2019, pp. 1201–1204.
- [417] Juanjuan Tu, Yongzhao Zhan, and Fei Han. “A neural network pruning method optimized with PSO algorithm.” In: *2010 Second International Conference on Computer Modeling and Simulation*. Vol. 3. IEEE. 2010, pp. 257–259.
- [418] Ledyard R Tucker. “Some mathematical notes on three-mode factor analysis.” In: *Psychometrika* 31.3 (1966), pp. 279–311.

- [419] Frederick Tung and Greg Mori. “Similarity-preserving knowledge distillation.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 1365–1374.
- [420] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “Well-read students learn better: On the importance of pre-training compact models.” In: *arXiv preprint arXiv:1908.08962* (2019).
- [421] Joseph Turian, Lev Ratinov, and Yoshua Bengio. “Word representations: a simple and general method for semi-supervised learning.” In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics. 2010, pp. 384–394.
- [422] Amos Tversky. “Features of similarity.” In: *Psychological review* 84.4 (1977), p. 327.
- [423] Karen Ullrich, Edward Meeds, and Max Welling. “Soft weight-sharing for neural network compression.” In: *arXiv preprint arXiv:1702.04008* (2017).
- [424] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need.” In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.
- [425] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. “Cider: Consensus-based image description evaluation.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4566–4575.
- [426] Andreas Veit, Michael J Wilber, and Serge Belongie. “Residual networks behave like ensembles of relatively shallow networks.” In: *Advances in neural information processing systems*. 2016, pp. 550–558.
- [427] Greg Ver Steeg. *Non-parametric entropy estimation toolbox (npeet)*. Tech. rep. Technical Report. 2000. Available online: <https://www.isi.edu/~gregv...>, 2000.
- [428] Greg Ver Steeg and Aram Galstyan. “Information-theoretic measures of influence based on content dynamics.” In: *Proceedings of the sixth ACM international conference on Web search and data mining*. 2013, pp. 3–12.
- [429] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. “Show and tell: A neural image caption generator.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3156–3164.
- [430] Hermann Von Schelling. “Coupon collecting for unequal probabilities.” In: *The American Mathematical Monthly* 61.5 (1954), pp. 306–311.

- [431] Weishui Wan, Shingo Mabu, Kaoru Shimada, Kotaro Hirasawa, and Jinglu Hu. “Enhancing the generalization ability of neural networks through controlling the hidden layers.” In: *Applied Soft Computing* 9.1 (2009), pp. 404–414.
- [432] Alex Wang, Amapreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding.” In: *arXiv preprint arXiv:1804.07461* (2018).
- [433] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. “Eigendamage: Structured pruning in the kronecker-factored eigenbasis.” In: *International Conference on Machine Learning*. PMLR, 2019, pp. 6566–6575.
- [434] Dongdong Wang, Yandong Li, Liqiang Wang, and Boqing Gong. “Neural Networks Are More Productive Teachers Than Human Raters: Active Mixup for Data-Efficient Knowledge Distillation from a Blackbox Model.” In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1498–1507.
- [435] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. “Training deep neural networks with 8-bit floating point numbers.” In: *Advances in neural information processing systems*. 2018, pp. 7675–7684.
- [436] Xiaojie Wang, Rui Zhang, Yu Sun, and Jianzhong Qi. “Kdgan: Knowledge distillation with generative adversarial networks.” In: *Advances in Neural Information Processing Systems*. 2018, pp. 775–786.
- [437] Colin Wei, Jason Lee, Qiang Liu, and Tengyu Ma. “On the margin theory of feedforward neural networks.” In: *OpenReview* (2018).
- [438] Andreas S Weigend, David E Rumelhart, and Bernardo A Huberman. “Generalization by weight-elimination with application to forecasting.” In: *Advances in neural information processing systems*. 1991, pp. 875–882.
- [439] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. “Feature hashing for large scale multitask learning.” In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 1113–1120.
- [440] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. “Learning structured sparsity in deep neural networks.” In: *Advances in neural information processing systems*. 2016, pp. 2074–2082.

- [441] Darrell Whitley, Timothy Starkweather, and Christopher Bogart. “Genetic algorithms and neural networks: Optimizing connections and connectivity.” In: *Parallel computing* 14.3 (1990), pp. 347–361.
- [442] Ronald J Williams and David Zipser. “A learning algorithm for continually running fully recurrent neural networks.” In: *Neural computation* 1.2 (1989), pp. 270–280.
- [443] Genta Indra Winata, Zhaojiang Lin, and Pascale Fung. “Learning multilingual meta-embeddings for code-switching named entity recognition.” In: *Proc. of RepL4NLP*. 2019, pp. 181–186.
- [444] Genta Indra Winata, Zhaojiang Lin, Jamin Shin, Zihan Liu, and Pascale Fung. “Hierarchical meta-embeddings for code-switching named entity recognition.” In: *arXiv preprint arXiv:1909.08504* (2019).
- [445] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Zhicheng Yan, Masayoshi Tomizuka, Joseph Gonzalez, Kurt Keutzer, and Peter Vajda. “Visual transformers: Token-based image representation and processing for computer vision.” In: *arXiv preprint arXiv:2006.03677* (2020).
- [446] Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. “Sampling matters in deep embedding learning.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2840–2848.
- [447] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. “Quantized convolutional neural networks for mobile devices.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4820–4828.
- [448] Mike Wu, Chengxu Zhuang, Milan Mosse, Daniel Yamins, and Noah Goodman. “On Mutual Information in Contrastive Learning for Visual Representations.” In: *arXiv preprint arXiv:2005.13149* (2020).
- [449] Qi Wu, Chunhua Shen, Peng Wang, Anthony Dick, and Anton van den Hengel. “Image captioning and visual question answering based on attributes and external knowledge.” In: *IEEE transactions on pattern analysis and machine intelligence* 40.6 (2018), pp. 1367–1381.
- [450] Shijie Wu, Alexis Conneau, Haoran Li, Luke Zettlemoyer, and Veselin Stoyanov. “Emerging cross-lingual structure in pretrained language models.” In: *arXiv preprint arXiv:1911.01464* (2019).
- [451] Xin Wu, Yi Cai, Yang Kai, Tao Wang, and Qing Li. “Task-oriented Domain-specific Meta-Embedding for Text Classification.” In: *Proc. of EMNLP*. 2020, pp. 3508–3513.

- [452] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. “Un-supervised feature learning via non-parametric instance discrimination.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3733–3742.
- [453] Tong Xiao, Shuang Li, Bochao Wang, Liang Lin, and Xiaogang Wang. “Joint detection and identification feature learning for person search.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3415–3424.
- [454] Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. “Sharing attention weights for fast transformer.” In: *arXiv preprint arXiv:1906.11024* (2019).
- [455] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. “Aggregated residual transformations for deep neural networks.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500.
- [456] Yuqiang Xie, Yue Hu, Luxi Xing, and Xiangpeng Wei. “Dynamic task-specific factors for meta-embedding.” In: *International Conference on Knowledge Science, Engineering and Management*. Springer. 2019, pp. 63–74.
- [457] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. “Show, attend and tell: Neural image caption generation with visual attention.” In: *International Conference on Machine Learning*. 2015, pp. 2048–2057.
- [458] Jian Xue, Jinyu Li, and Yifan Gong. “Restructuring of deep neural network acoustic models with singular value decomposition.” In: *Interspeech*. 2013, pp. 2365–2369.
- [459] Jian Xue, Jinyu Li, Dong Yu, Mike Seltzer, and Yifan Gong. “Singular value decomposition based low-footprint speaker adaptation and personalization for deep neural network.” In: *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2014, pp. 6359–6363.
- [460] Chenglin Yang, Lingxi Xie, Siyuan Qiao, and Alan L Yuille. “Training deep neural networks in generations: A more tolerant teacher educates better students.” In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 5628–5635.
- [461] Yinfei Yang, Yuan Zhang, Chris Tar, and Jason Baldridge. “PAWS-X: A cross-lingual adversarial dataset for paraphrase identification.” In: *Empirical Methods in Natural Language Processing*. 2019.
- [462] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. “Breaking the softmax bottleneck: A high-rank RNN language model.” In: *arXiv preprint arXiv:1711.03953* (2017).

- [463] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. “XLNet: Generalized Autoregressive Pretraining for Language Understanding.” In: *arXiv preprint arXiv:1906.08237* (2019).
- [464] Amir Yazdanbakhsh, Ahmed T Elthakeb, Pranoy Pilligundla, FatemehSadat Mireshghallah, and Hadi Esmaeilzadeh. “Releq: An automatic reinforcement learning approach for deep quantization of neural networks.” In: *arXiv preprint arXiv:1811.01704* (2018).
- [465] Jianbo Ye, Xin Lu, Zhe Lin, and James Z Wang. “Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers.” In: *arXiv preprint arXiv:1802.00124* (2018).
- [466] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. “Learning compact recurrent neural networks with block-term tensor decomposition.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9378–9387.
- [467] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. “A gift from knowledge distillation: Fast optimization, network minimization and transfer learning.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4133–4141.
- [468] Wenpeng Yin and Hinrich Schütze. “Learning meta-embeddings by using ensembles of embedding sets.” In: *arXiv preprint arXiv:1508.04257* (2015).
- [469] Wenpeng Yin and Hinrich Schütze. “Learning word meta-embeddings.” In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2016, pp. 1351–1360.
- [470] Zi Yin and Yuanyuan Shen. “On the dimensionality of word embedding.” In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 895–906.
- [471] Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Xiaohan Chen, Yingyan Lin, Zhangyang Wang, and Richard G Baraniuk. “Drawing early-bird tickets: Towards more efficient training of deep networks.” In: *arXiv preprint arXiv:1909.11957* (2019).
- [472] Peter Young, Alice Lai, Micah Hodosh, and Julia Hockenmaier. “From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions.” In: *Transactions of the Association for Computational Linguistics 2* (2014), pp. 67–78.

- [473] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. “SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient.” In: *AAAI*. 2017, pp. 2852–2858.
- [474] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. “Long-term forecasting using tensor-train rnns.” In: *Arxiv* (2017).
- [475] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. “Nisp: Pruning networks using neuron importance score propagation.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9194–9203.
- [476] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. “On compressing deep models by low rank and sparse decomposition.” In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 7370–7379.
- [477] Ming Yuan and Yi Lin. “Model selection and estimation in regression with grouped variables.” In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.1 (2006), pp. 49–67.
- [478] Sergey Zagoruyko and Nikos Komodakis. “Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer.” In: *arXiv preprint arXiv:1612.03928* (2016).
- [479] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks.” In: *BMVC*. 2016.
- [480] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks.” In: *arXiv preprint arXiv:1605.07146* (2016).
- [481] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. “Barlow twins: Self-supervised learning via redundancy reduction.” In: *arXiv preprint arXiv:2103.03230* (2021).
- [482] Chiyuan Zhang, Samy Bengio, and Yoram Singer. “Are All Layers Created Equal?” In: *arXiv preprint arXiv:1902.01996* (2019).
- [483] Dejian Zhang, Haozhu Wang, Mario Figueiredo, and Laura Balzano. “Learning to share: Simultaneous parameter tying and sparsification in deep learning.” In: *OpenReview.net* (2018).
- [484] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. “mixup: Beyond empirical risk minimization.” In: *arXiv preprint arXiv:1710.09412* (2017).
- [485] Li Zhang, Flood Sung, Feng Liu, Tao Xiang, Shaogang Gong, Yongxin Yang, and Timothy M Hospedales. “Actor-critic sequence training for image captioning.” In: *arXiv preprint arXiv:1706.09601* (2017).

- [486] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. “Residual dense network for image super-resolution.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2472–2481.
- [487] Mengjie Zhao, Tao Lin, Martin Jaggi, and Hinrich Schütze. “Masking as an Efficient Alternative to Finetuning for Pretrained Language Models.” In: *arXiv preprint arXiv:2004.12406* (2020).
- [488] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. “Incremental network quantization: Towards lossless cnns with low-precision weights.” In: *arXiv preprint arXiv:1702.03044* (2017).
- [489] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. “Explicit loss-error-aware quantization for low-bit deep neural networks.” In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 9426–9435.
- [490] Chunting Zhou, Graham Neubig, and Jiatao Gu. “Understanding Knowledge Distillation in Non-autoregressive Machine Translation.” In: *arXiv preprint arXiv:1911.02727* (2019).
- [491] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. “Deconstructing lottery tickets: Zeros, signs, and the supermask.” In: *Advances in Neural Information Processing Systems*. 2019, pp. 3597–3607.
- [492] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients.” In: *arXiv preprint arXiv:1606.06160* (2016).
- [493] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. “Trained ternary quantization.” In: *arXiv preprint arXiv:1612.01064* (2016).
- [494] Michael Zhu and Suyog Gupta. “To prune, or not to prune: exploring the efficacy of pruning for model compression.” In: *arXiv preprint arXiv:1710.01878* (2017).
- [495] Xiatian Zhu, Shaogang Gong, et al. “Knowledge distillation by on-the-fly native ensemble.” In: *Advances in neural information processing systems*. 2018, pp. 7517–7527.
- [496] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. “Texygen: a benchmarking platform for text generation models.” In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. ACM. 2018, pp. 1097–1100.
- [497] Chengxu Zhuang, Alex Lin Zhai, and Daniel Yamins. “Local aggregation for unsupervised learning of visual embeddings.” In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 6002–6012.



## COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst’s seminal book on typography “*The Elements of Typographic Style*”. `classicthesis` is available for both L<sup>A</sup>T<sub>E</sub>X and L<sup>Y</sup>X:

<https://bitbucket.org/amiede/classicthesis/>

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

<http://postcards.miede.de/>

Thank you very much for your feedback and contribution.

*Final Version* as of July 1, 2022 (`classicthesis` v4.6).



# A

---

## APPENDIX

---

### A.1 FURTHER DETAILS ON LAYER FUSION

**MACHINE SPECIFICATION.** We use an ASUS ROG Strix GL503VS SCAR gaming laptop with an NVIDIA GTX 1070 8GB Graphics Processing Unit (GPU) card for training all models.

**FUSING LAYERS OF UNEQUAL SIZE** For a pair of vectorized tensors of column size  $d$  and  $d + k$ , we remove  $k$  weights that have the smallest magnitudes from the  $2^{nd}$  tensor until both match. We also considered using PCA and SVD to fix the dimension for each layer pair. However, this is less of an issue in the case of convolutional layers since filters that are most similar tend to be in the same layer. For Transformer models, the same parts of each attention block are fused e.g the key weight tensor from one layer could not be fused with a value weight tensor from another, only another key weight tensor, and these are the same dimensions, hence the same length when vectorized.

**APPROXIMATING THE COVARIANCE MATRIX** For our experiments, some of the layers can be relatively large. For example, the large GPT-2 a weight matrix from a given hidden layer is  $\mathbf{w} \in \mathbb{R}^{2048 \times 2048}$  and a total of 4,194,304 parameters. We split  $\mathbf{W}$  into block matrices to perform covariance estimation. The row  $d_r$  and column  $d + c$  dimensionality are restricted to  $d_r \leq d_r \leq 128$ . A submatrix  $\mathbf{w}_{i,j}$  can be represented as,

$$\mathbf{w}_{i,j} = \begin{bmatrix} \mathbf{w}_{1,1} & \dots & \mathbf{w}_{1,d_c} \\ \vdots & \ddots & \\ \mathbf{w}_{d_r,1} & & \mathbf{w}_{d_r,d_c} \end{bmatrix}$$

and all submatrices of  $\mathbf{W} \in \mathbb{R}^{n \times m}$  are then formed where  $m = \text{round}(d_r/128)$  and  $n = \text{round}(d_c/128)$ . Given a pair of submatrices from two adjacent layers  $\mathbf{w}_{i,j}^{n_\ell} \subset \mathbf{W}^{n_\ell}$  and  $\mathbf{w}_{i,j}^{n_{\ell+1}} \subset \mathbf{W}^{n_{\ell+1}}$  from layers  $n_\ell$  and  $n_{\ell+1}$  respectively, we estimate the covariance similarity between them. This is computed for all adjacent  $m, n$  submatrix pairs, assuming

that  $d_r^{n_\ell} = d_r^{n_{\ell+1}}$  and  $d_c^{n_\ell} = d_c^{n_{\ell+1}}$ . The complete covariance similarity is then estimated by its mean as,

$$\mathbb{E}[\Sigma_{\mathbf{w}}] = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n d_{cov}(\Sigma_{\mathbf{w}_{i,j}^{n_\ell}}, \Sigma_{\mathbf{w}_{i,j}^{n_{\ell+1}}}) \quad (\text{A.1})$$

The covariance estimation techniques are then applied to compute covariance matrix similarity.

## A.2 CONDITIONAL NEGATIVE SAMPLING

### A.2.1 *Hardware Details*

All experiments were run on a Titan RTX P8 24G memory graphics processing unit.

### A.2.2 *Network Architectures*

The below CNN architectures are used for standard supervised learning experiments on CIFAR10 and for KD experiments on CIFAR100 and Tiny-ImageNet-200.

- Wide Residual Network [WRN; 480]. WRN-d-w represents wide resnet with depth d and width factor w.
- ResNet [151]. We use ResNet-d to represent cifar-style resnet with 3 groups of basic blocks, each with 16, 32, and 64 channels respectively. In our experiments, resnet8x4 and resnet32x4 indicate a 4 times wider network (namely, with 64, 128, and 256 channels for each of the block).
- ResNet [151]. ResNet-d represents ImageNet-style ResNet with Bottleneck blocks and more channels.
- ResNeXt [455]. ResNeXt ImageNet-style ResNet with Bottleneck blocks and more channels.
- VGG [392]. The pretrained VGG network are adapted from its original ImageNet counterpart.
- DenseNet [174]. We use a pretrained ImageNet DenseNet-121 and fine-tune on Tiny-ImageNet-200 with upscale images (64x64 to 256x256).

### A.2.3 *Experiment Details*

#### A.2.3.1 *Conditional Negative Sampling Details*

Before running experiments for supervised learning and knowledge distillation, we must first define the negative sampling distribution on the instance-level or class-level. For class-level SCNS we use cross-modal transfer by using word embeddings for the class labels. We use

Abbreviation	KD Method
KLD	Knowledge Distillation [KD; 162] - <b>Kullbeck Leibler Divergence</b>
Fitnets	<b>Fitnets</b> : Hints for thin deep nets [360]
AT	<b>Attention Transfer</b> [AT; 478]
SP	<b>Similarity-Preserving</b> Knowledge Distillation [SP; 419]
CC	<b>Correlation Congruence</b> [CC; 333]
VID	<b>Variational information distillation</b> for knowledge transfer [VID; 4]
RKD	<b>Relational Knowledge Distillation</b> [RKD; 328]
PKT	Learning deep representations with <b>probabilistic knowledge transfer</b> [PKT; 330]
FT	Paraphrasing complex network: Network compression via <b>factor transfer</b> [FT; 198]
FSP	A gift from knowledge distillation: Fast optimization, network minimization and transfer learning [FSP; 467]

Table A.1: Abbreviations for Knowledge Distillation Baselines

**skipgram** word vectors [283] that are pretrained on GoogleNews and can be retrieved from <https://code.google.com/archive/p/word2vec/>. For class labels that are phrases, we average the pretrained word embeddings of each constituent embedding prior to computing cosine similarity. We find best results for our proposed method with the temperature  $\tau = 5$  when constructing  $\mathbf{P}$ . This ensures that the distribution is not too flat and encourages tighter coupling of neighbours.

For instance-level SCNS, pair similarity is defined by a pretrained network of the same type that is used for training in the supervised learning setting. For knowledge distillation, the teacher network is used to define the pair similarity.

#### A.2.3.2 Dataset and Model Details

For all models used in the standard supervised learning and KD settings, we use the cross-entropy loss optimized using Stochastic Gradient Descent (SGD) with a decay rate (different setting for each task). Additionally, hyperparameter tuning of  $\beta$  and  $k$  is tested on a randomly sampled 5% of the predefined training data of all three datasets.

**CIFAR-10** For CIFAR-10, the learning rate is set to 0.01, momentum=0.9 and weight\_decay=0.0005. The images are randomly cropped and horizontally flipped and normalized along the input channels (as two tuple arguments (0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010) in the `transforms.Normalize` method in the `torchvision` library). The batch size is 128 for training.

**CIFAR-100** For CIFAR-100 a dataset with 50k training images (500 samples per 100 classes) and 10k test images, the learning rate is set to 0.05 with a decay rate 0.1 at every 25 epochs after 100 epochs until the last 200 epoch. The batch size is set to 64.

**Tiny-ImageNet-200** For Tiny-ImageNet-200, we train for 100 epochs and decay the learning rate every 20 epochs. The batch size is also set to 64. The student is trained by a combination of cross-entropy classification objective and a KD objective as  $\ell = \ell_{CE} + \alpha\ell_{KD}$ .

**BASELINE KD SETTINGS** The abbreviations refer to correspond to the KD method names listed in Table A.1.

The main KD influence factor  $\alpha$  is set based on either the original paper settings or a set using a grid search over few settings close to the original paper settings. For our proposed Pearson Correlation (PC) and Centered Kernel Alignment (CKA) KD objectives, we grid search over  $\alpha \in [0.2, 0.5, 0.65, 0.8, 0.9]$  and  $\zeta \in [0.1, 0.2, 0.5, 0.7, 0.9]$  for those objectives that use a margin-based triplet loss (e.g Triplet CKA). The parameter settings specified in the paper of previous KD methods is used and where not specified we manually grid search different settings of  $\alpha$ .

1. Fitnets [360]:  $\alpha = 80$
2. AT [478]:  $\alpha = 600$
3. SP [419]:  $\alpha = 2000$
4. CC [333]:  $\alpha = 0.05$
5. VID [4]:  $\alpha = 0.8$
6. RKD [328]:  $\alpha_1 = 25$  for distance and  $\alpha_2 = 50$  for angle. For this loss, we combine both term following the original paper.
7. PKT Probabilistic Knowledge Transfer [330] [330]:  $\alpha = 10000$
8. AB [157]:  $\alpha = 0.2$ , distillation happens in a separate pre-training stage where only distillation objective applies.
9. FT [198]:  $\alpha = 500$
10. FSP [467]:  $\alpha = 0$ , distillation happens in a separate pre-training stage where only distillation objective applies.
11. NST [176]:  $\alpha = 50$
12. CRD Contrastive Representation Distillation [414]:  $\alpha = 0.8$ , in general  $\alpha \in [0.5, 1.5]$  works well.
13. Kullbeck-Leibler Divergence Distillation (KLD) [162]:  $\alpha = 0.9$  and  $\tau = 4$ .

#### OUR PROPOSED KD BASELINE METHOD SETTINGS

1. Siamese-PC: This is the loss from Equation A.5.  $\alpha = 0.8$
2. Triplet CKA (Linear) or referred to as Linear-CKA: This is the loss from Equation A.4 loss with a linear kernel -  $\alpha = 0.8$  and  $\zeta = 0.2$
3. Triplet CKA (RBF) or referred to as Linear-RBF:  $\alpha = 0.8$  and  $\zeta = 0.2$
4. Contrastive-CKA described in Equation A.4:  $\alpha = 0.15$
5. Contrastive-PC: This is the loss in Equation A.5 applied to both the positive pair embeddings and negative sample pair embeddings.

#### OUR PROPOSED SCNS-BASED KD SETTINGS

1. InfoNCE Loss with Instance-level SCNS:  $\alpha = 0.9$
2. InfoNCE-LM: This is the InfoNCE loss between latent mixup representations as in Equation 9.

3. InfoNCE+Class SCNS: This represents SCNS with an InfoNCE loss.  $\alpha = 0.65$
4. InfocNCE-LM + Class-SCNS: This represents SCNS with an InfoNCE loss with an second InfoNCE loss for latent mixup representations.  $\alpha = 0.65$

**PREPROCESSING DETAILS** For experiments with the CKA objective we group mini-batches by their targets as CKA operates on cross-correlations between samples of the same class. Therefore, random shuffling is carried out on the mini-batch level but not on the instance level. For all other objectives, standard random shuffling of the training data is performed.

#### A.2.4 Metric Learning Distillation Objectives

In our work we also propose two correlation and kernel-based loss functions that can be used for both standard pointwise-based KD and metric-learned KD. These are used as alternatives from those described in the related research, which we describe below.

**METRIC-BASED CENTERED KERNEL ALIGNMENT** The CKA function measures the closeness of two set of points that can be represented as matrices. Thus far it has only been used for analysing representation similarity in neural networks [206] but not for optimizing a neural network. We propose to distil the knowledge of the teacher network by minimizing the alignment between student and teacher representations using CKA as a baseline.

For two arbitrary matrices  $\mathbf{Z}_i \in \mathbb{R}^{M \times d_L}$  and  $\mathbf{Z}_j \in \mathbb{R}^{M \times d_L}$ , each consisting of a set of neural network representations, the centered alignment (CA) can be expressed as,

$$\text{CA}(\mathbf{Z}_i, \mathbf{Z}_j) = \frac{\langle \text{vec}(\mathbf{Z}_i \mathbf{Z}_i^\top), \text{vec}(\mathbf{Z}_j \mathbf{Z}_j^\top) \rangle}{\|\mathbf{Z}_i \mathbf{Z}_i^\top\|_F \|\mathbf{Z}_j \mathbf{Z}_j^\top\|_F} \quad (\text{A.2})$$

where  $\|\cdot\|_F$  is the Frobenius norm. We can replace the the dot product in numerator with a kernel function  $K(\cdot, \cdot)$  to compute the CKA. The kernel function is smooth and differentiable, hence we use it as a loss function for KD-based metric learning to maximize the similarity between the positive class latent representations given by  $(z_+^S, z_+^T)$  and negative class latent representations  $(z_-^S, z_-^T)$ . Equation A.3 shows the formulation of CKA where a kernel is used instead of the dot product.

$$\begin{aligned} \text{CKA}(\mathbf{Z}_i, \mathbf{Z}_j) = & K(\mathbf{Z}_i, \mathbf{Z}_j) - \mathbb{E}_{\mathbf{Z}}[K(\mathbf{Z}_i, \mathbf{Z}_j)] - \\ & \mathbb{E}_{\mathbf{Z}_j}[K(\mathbf{Z}_i, \mathbf{Z}_j)] + \mathbb{E}_{\mathbf{Z}_i, \mathbf{Z}_j}[K(\mathbf{Z}_i, \mathbf{Z}_j)] \end{aligned} \quad (\text{A.3})$$

In our experiments, a linear kernel and a radial basis function ( $K(\mathbf{Z}_i, \mathbf{Z}_j) = \exp(-\|\text{vec}(\mathbf{Z}_i) - \text{vec}(\mathbf{Z}_j)\|_F^2 / 2S^2)$ ) were used where  $S^2$  is the sample variance. To account for intra-variations and inter-variations between student and teacher representations the CKA loss is used as part of a triplet loss that maximizes the kernel similarity between the positive pair of the student anchor and student positive sample and also the student anchor with the teacher anchor. This is shown as  $\ell_{\text{CKA}}^+$  in Equation A.4, where  $z_*$  represents the anchor sample. The same is computed for the negative pair, denoted by  $\ell_{\text{CKA}}^-$ . Both losses are combined as one where  $\zeta$  controls the tradeoff between positive pair losses and negative pair losses and  $m$  is the margin.

$$\begin{aligned}\ell_{\text{CKA}}^+ &= \text{CKA}(z_+^S, z_*^S) + \text{CKA}(z_*^S, z_*^T), \\ \ell_{\text{CKA}}^- &= \text{CKA}(z_-^S, z_*^S) + \text{CKA}(z_-^S, z_*^T), \\ \ell_{\text{CKA}} &= \max\left(0, \zeta \ell_{\text{CKA}}^+ - (1 - \zeta) \ell_{\text{CKA}}^- + m\right)\end{aligned}\tag{A.4}$$

Concretely, this will force all positive class representations to have high a CKA score within and across the samples for the student and teacher representations, and similarly for the negative pair of the triplet.

**PEARSON CORRELATION REPRESENTATION DISTILLATION** An alternative to maximizing the mutual information between  $z^S$  and  $z^T$  [27] is instead to maximize the linear interactions using a PC-based loss as a strong baseline. The objective to be maximized is expressed as Equation A.5

$$\begin{aligned}\ell_{\text{PC}}^+ &= \rho_{\text{PC}}(z_-^S, z_-^T) + \rho_{\text{PC}}(z_+^S, z_+^T), \\ \ell_{\text{PC}}^- &= \rho_{\text{PC}}(z_-^S, z_+^S) + \rho_{\text{PC}}(z_-^S, z_+^T), \\ \ell_{\text{PC}} &= \max\left(0, \zeta \ell_{\text{PC}}^+ - (1 - \zeta) \ell_{\text{PC}}^- + m\right)\end{aligned}\tag{A.5}$$

where  $\rho_{\text{PC}} \in [-1, 1]$  computes the correlation coefficient. When using the  $\ell_{\text{PC}}$  loss with contrastive learning ('Contrastive-PC') we take the average loss as  $\frac{1}{N-1} \sum_{i=1}^{N-1} \rho_{\text{PC}}(z_{-,i}^S, z_+^S)$  and similarly for the remaining losses that use negative sample representations.

#### A.2.5 Connection Between Mutual Information & Conditional Negative Sampling

In this section we describe contrastive learning with our proposed conditional negative sampling in terms of mutual information (MI). Let  $p(y)$  be the probability of observing the class label  $y$  and  $p(x, y)$  denote the probability density function of the corresponding joint distribution. Then, the MI is defined as Equation A.6

$$I(X; Y) = \sum_y \int_x p(x, y) \log \frac{p(x, y)}{p(x)p(y)} dx\tag{A.6}$$

and can be further expressed in terms of the entropy  $\mathbb{H}(X)$  and conditional entropy  $\mathbb{H}(X|Y)$  as shown in [Equation A.7](#).

$$\begin{aligned}
I(X; Y) &= \sum_y \int_x p(x, y) \log \frac{p(x|y)}{p(x)} dx \\
&= - \int_x p(x) \log p(x) - \left( - \int_x \sum_y \log p(x, y) p(x|y) \right) \\
&= \mathbb{H}(X) - \mathbb{H}(X|Y)
\end{aligned} \tag{A.7}$$

Then  $I(X; Y)$  can be formulated as the KL divergence between  $p(x, y)$  and the product of marginals  $p(x)$  and  $p(y)$ ,

$$I(X; Y) = D_{KL}(p(x, y) || p(x)p(y)) = \mathbb{E}_{p(x, y)} \left[ \frac{p(x, y)}{p(x)p(y)} \right] \tag{A.8}$$

Hence, if the classifier can accurately distinguish between samples drawn from the joint  $p(x, y)$  and those drawn from the product of marginals  $p(x)p(y)$ , then  $X$  and  $Y$  have a high MI. However, estimating MI between high-dimensional continuous variables is difficult and therefore easier to approximate by maximizing a lower bound on MI. This is known as the InfoMax principle [246]. In the below subsections, we describe how this MI lower bound is maximized using the InfoNCE loss [321].

#### A.2.5.1 Estimating Mutual Information with InfoNCE

The InfoNCE loss maximizes the MI between  $\mathbf{z}_i$  and  $\mathbf{z}_j$  (which is bounded by the MI between  $\mathbf{z}_i$  and  $\mathbf{z}_j$ ). The optimal value for  $f(\mathbf{z}_j, \mathbf{z}_i)$  is given by  $p(\mathbf{z}_j|\mathbf{z}_i)/p(\mathbf{z}_j)$ . Inserting this back into Equation 4 and splitting  $X$  into the positive sample and the negative examples  $X_-$  results in:

$$\begin{aligned}
\ell &= -\mathbb{E}_X \log \left[ \frac{\frac{p(x_*|x_i)}{p(x_*)}}{\frac{p(x_*|x_+)}{p(x_*)} + \sum_{x_i \in X_-} \frac{p(x_i|x_+)}{p(x_i)}} \right] = \\
&\mathbb{E}_X \log \left[ 1 + \frac{p(x_*)}{p(x_*|x_+)} + \sum_{x_i \in X_-} \frac{p(x_i|x_+)}{p(x_i)} \right] \\
&\approx \mathbb{E}_X \log \left[ 1 + \frac{p(x_*)}{p(x_*|x_+)} (M-1) \mathbb{E}_{x_i} \frac{p(x_i|x_+)}{p(x_i)} \right] \\
&= \mathbb{E}_X \log \left[ 1 + \frac{p(x_*)}{p(x_*|x_+)} (M-1) \right] \\
&\geq \mathbb{E}_X \log \left[ \frac{p(x_*)}{p(x_*|x_+)} M \right] \\
&= -I(x_*, x_+) + \log(M)
\end{aligned} \tag{A.9}$$

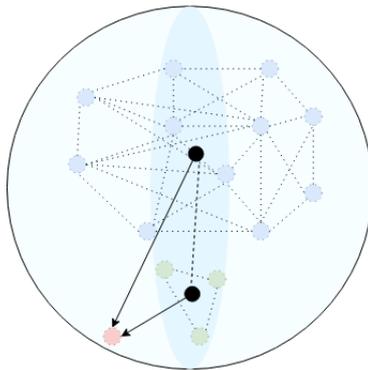


Figure A.1:  $k$ -NN Conditional Negative Sampling for embeddings on the  $L_2$  sphere retrieved from the lookup table.

Therefore,  $I(z_j, z_i) \geq \log(N) - \ell$  which holds for any  $f$ , where higher  $\ell$  leads to a looser MI bound. This MI bound becomes tighter as the number of negative sample pairs  $M$  increases and in turn is likely to reduce  $\ell$ . In our work we argue that the  $\log(M)$  term be replaced with a term that accounts for the geometry of the embedding space as not all negative sample pairs are equally important for reducing  $\ell$ . Therefore, the next subsection describes *our* formulation of the MI bound that incorporates the notion of semantic similarity between embeddings of the sample pairs in order to choose an informative  $M$  samples to tighten the bound.

#### A.2.5.2 Mutual Information Lower Bounds for Conditioned Negative Sampling

We formalize the connection between SCNS and maximizing MI between representations in the contrastive learning and formulate an expression that describes how SCNS tightens the lower bound estimate on MI. We begin by defining ‘closeness’ between representations  $\{z_*, z_+, z_-\} \in \mathcal{Z}$  of samples  $\{x_*, x_+, x_-\} \in \mathcal{X}$ . Given that we are not restricted to a distance, divergence or similarity between vectors, we refer to a general measure as an *alignment* function  $A$  that outputs an alignment score  $a \in [0, 1]$ .

Given an anchor sample  $x_*$ , the expected alignment score for the top- $k$  negative samples is  $a_{x_-}^k := \mathbb{E}_{x_- \sim D_x^k}[A(z_*, z_-)]$  and for the negative samples outside of the top- $k$  samples it is  $a_{x_-}^r := \mathbb{E}_{x_- \sim D_x^r}[A(z_*, z_-)]$  where  $D_{x_*}^r \subseteq D$ ,  $D_{x_*}^k \not\subseteq D_{x_*}^r$ ,  $r = N - N_y - k$ . and  $N_y$  is the number of samples of class  $y$ .

From the above,  $\alpha \approx 1$  corresponds to negative samples that lie very close to  $x_*$ . We can then use the alignment weight (AW)  $\Omega_{x_*} := 1 - a_{x_*}^k / (a_{x_*}^k + a_{x_*}^r)$  to represent the difference in ‘closeness’ between the top- $k$  negative samples and the remaining negative samples. This is visualized as the difference in alignment between the centroids in the embedding space shown in Figure A.1. Blue circles are *easy* negative

sample embeddings, green circles are the top- $k$  most semantically similar embeddings for the target sample in green and black circles represent embedding centroids computed using a weighted average of embedding where the weights are the alignment score.

We can then replace  $M$  negative samples as  $\Omega := \sum_{i=1}^M \Omega_{x_i}$  and substitute  $\log(M)$  with  $\log(2\Omega)$  in Equation A.9. When the centroids of both negative samples sets are close (i.e.  $a_{x_*}^k = a_{x_*}^r$ ) then  $\Omega = M/2$ . Hence  $\log(2\Omega) \approx \log(M)$  when the negative samples are centered in the same region, in which case uniform sampling provides the same guarantees as SCNS. The new MI lower bound is then Equation A.10.

$$I(X, Y) \geq \ell + \log(2\Omega) \quad (\text{A.10})$$

Intuitively, this bound favors top- $k$  negative samples that are close to the positive class boundary but are also relatively close compared to the remaining negative samples. This is dependent on how the embedding space is constructed from the pretrained network and which  $A$  is chosen. Hence, this is clearly an estimation on the true MI lower bound.

We can now define the relation between  $M$  uniformly sampled negatives (USNs) and top- $k$  SCNSs. Let  $\Omega_{D_x^U}$  be the AW of USNs for  $x_*$  and  $\Omega_{D_x^k}$  is the AW for the top- $k$  negative samples. When  $|D_{x_*}^U| \approx |D_{x_*}^k|$  (i.e both negative sample sets lie on a ring on the hypersphere around  $x_*$ ),

$$I(X, Y) \leq \ell + \log(2\Omega_{D_x^k}) \leq \ell + \log(2\Omega_U) \quad (\text{A.11})$$

given the non-uniform prior over negative samples as defined in SCNS. For some  $k \ll N$ ,  $2(\Omega_k - \Omega_U) = 0$  is met when a subset of  $D_x^U$  negative samples have  $\bar{z}_{\tilde{x}}^U \approx \bar{z}_x^k$  where the subscript  $\tilde{x}$  denotes the aforementioned subset and  $\bar{z}_x^k$  is the centroid of the top- $k$  negative samples

#### A.2.6 Uniform vs SCNS Sample Complexity

In this section, we aim to identify the relationship between uniform sampling and SCNS by formulating how many draws are required to cover the the top- $k$  samples. We begin by the simpler case of single draws with a uniform distribution and extend it to batches of negative samples of size  $b$  are drawn uniformly. We then repeat this with draws of unequal probability in Section A.2.6.2, as is the case for SCNS.

##### A.2.6.1 Number of Samples Until Observing Top- $k$ SCNS Samples Under a Uniform Distribution

We now describe the expected number of i.i.d drawn negative samples from  $M$  to observe the top- $k$  samples at least once for a given  $x$ . Let  $N_i$  denote the number of negative samples observed until you see

the  $i$ -th new negative sample among the top- $k$  samples and  $N$  is the number of samples until all top- $k$  negative samples are observed. Since  $N = \sum_{i=1}^k N_i$ ,

$$\mathbb{E}[N] = \mathbb{E}\left[\sum_{i=1}^k N_i\right] = \sum_{i=1}^k \mathbb{E}[N_i] \quad (\text{A.12})$$

where  $N_i$  follows a geometric distribution with parameter  $(k+1-i)/M$ . Therefore  $\mathbb{E}[N_i] = \frac{M}{k+1-i}$  and  $\mathbb{E}[N] = M \sum_{i=1}^k (k+1-i)^{-1} = M \sum_{i=1}^k i^{-1}$ .

NUMBER OF BATCHES TO COVER ALL SAMPLES In [Section A.2.6.1](#), we formulate the number of uniform negative samples required to cover the top- $k$  negative samples at least once for a single consecutive draws. However, in practice mini-batch training is carried out and therefore it is necessary to reformulate this for consecutive mini-batch draws of size  $b$  for a given  $x$  with replacement. This is a special case of the Coupon Collector's Problem.

$$\begin{aligned} \sum_{j=0}^{\infty} P(K > ib) &= \sum_{i=0}^{\infty} \left(1 - \frac{M!}{M^{ib}} \left\{ \begin{matrix} ib \\ M \end{matrix} \right\}\right) = \sum_{i=0}^{\infty} \left(1 - \frac{1}{M^{ib}} \sum_{j=0}^M (-1)^{M-j} \binom{M}{j} j^{ib}\right) \\ &= \sum_{j=0}^{\infty} \sum_{j=0}^{M-1} (-1)^{M-j+1} \binom{M}{j} \binom{j}{M}^{ib} = \sum_{j=0}^{M-1} (-1)^{M-j+1} \binom{M}{j} \frac{1}{1 - (\frac{j}{M})^b} \end{aligned} \quad (\text{A.13})$$

Hence, as  $M$  grows more mini-batch updates are needed until the top- $k$  *hard* negative samples are observed. Thus,  $b$  is required to be larger which is typically required in the MI formulation of NCE. To define the difference between USNs and SCNS we also need to define the CCP for unequal probabilities as defined by  $\mathbf{P}_{x_*}$  for  $x_*$ . We first make a distributional assumption. Here, we assume that the distances (or *alignment*) for  $x$  to its  $N$  negative samples follows a power law distribution. This is well-established for text [\[38, 342\]](#) and we also observe a power law trend when computing the cosine similarities between all pairs with  $f^T$ .

#### A.2.6.2 Number of Samples Until Observing Top- $k$ SCNS Samples Under an SCNS Distribution

In this subsection, we formulate the expected number of negative samples required for non-uniform sampling distributions, namely our proposed SCNS distribution provided by the teacher network.

MAXIMUM-MINIMUM IDENTITY APPROACH The number of draws required to observe all top- $k$  NS is  $C = \max\{C_1, \dots, C_N\}$  where  $N_i$  has a conditional probability  $p_i$  of being sampled as defined in SCNS. Since

the minimum of  $N_i$  and  $N_j$  is the number of negative samples needed to obtain either the  $i$ -th top- $k$  sample or the  $j$ -th top- $k$  sample, it follows that for  $i \neq j$ ,  $\min(N_i, N_j)$  has probability  $p_i + p_j$  and the same is true for the minimum of any finite number of these random variables. The Maximum-Minimums Identity [361] is then used to compute the expected number of draws:

$$\begin{aligned} \mathbb{E}[N] &= \mathbb{E}[\max_{i=1, \dots, M} N_i] = \sum_i \mathbb{E}[N_i] - \sum_{i < j} \mathbb{E}[\min(N_i, N_j)] \\ &+ \sum_{i < j < k} \mathbb{E}[\min(N_i, N_j, N_k)] - \dots + (-1)^{M+1} \mathbb{E}[\min(N_1, N_2, \dots, N_M)] \end{aligned} \quad (\text{A.14})$$

We can then express the above in terms of the individual probabilities associated with drawing  $M$  negative samples conditioned on a given  $x_*$  as,

$$\begin{aligned} \mathbb{E}[N] &= \sum_i \frac{1}{p_i} - \sum_{i < j} \frac{1}{p_i + p_j} + \\ &\sum_{i < j < k} \frac{1}{p_i + p_j + p_k} + (-1)^{M+1} \frac{1}{p_1 + \dots + p_M} \end{aligned} \quad (\text{A.15})$$

Since  $\int_0^\infty e^{-px} dx = \frac{e^{-px}}{p} \Big|_{x=0}^{x=+\infty} = \frac{1}{p}$ , integrating gives

$$1 - \prod_{i=1}^N (1 - e^{-p_i x}) = \sum_i e^{-p_i x} = \sum_{i < j} e^{-(p_i + p_j)x} + \dots + (-1)^{N+1} e^{-(p_1 + \dots + p_N)x} \quad (\text{A.16})$$

Hence, we get a concise equivalent expression [109]:

$$\mathbb{E}[X_-] = \int_0^{+\infty} \left( 1 - \prod_{i=1}^N (1 - e^{-p_i x}) \right) dx \quad (\text{A.17})$$

The probability of sampling the  $i$ -th top- $k$  negative sample is  $p_i \geq 0$  such that  $p_1 + \dots + p_N = 1$ . To determine  $\mathbb{E}[N]$ , we first assume that the number of negative samples to draw  $t$  as  $X_-(t)$ , follows a Poisson distribution with parameter  $\lambda = 1$ . Let  $\mathbb{I}_i$  be the inter-arrival time between the  $(i-1)$ -th and the  $i$ -th negative sample draw:  $\mathbb{I}_i$  has exponential distribution with parameter  $\lambda = 1$ . Let  $Z_i$  be the time in which the  $i$ -negative sample arrives for the *first* time (hence  $Z_i \sim \exp(p_i)$ ) and let  $Z = \max\{Z_1, \dots, Z_N\}$  be the time in which we have observed all samples at least once. Note that  $Z = \sum_{i=1}^N \mathbb{I}_i$  and  $\mathbb{E}[X] = \mathbb{E}[Z]$ , indeed:

$$\begin{aligned}
\mathbb{E}[Z] &= \mathbb{E}[\mathbb{E}[Z|N]] = \sum_k \mathbb{E}\left[\sum_{i=1}^k \mathbb{I}_i | N = k\right] P(N = k) \\
&= \sum_k \left[\sum_{i=1}^k \mathbb{I}_i\right] \mathbb{P}(N = k) = \sum_k \sum_{i=1}^k \mathbb{E}[\mathbb{I}_i] \mathbb{P}(N = k) \\
&\quad \sum_k k \mathbb{P}(N = k) = \mathbb{E}(N)
\end{aligned} \tag{A.18}$$

It follows that it suffices to calculate  $\mathbb{E}[Z]$  to get  $\mathbb{E}[N]$ . Since  $Z = \max\{Z_1, \dots, Z_N\}$ , we have  $F_Z(t) = \mathbb{P}(Z \leq t) = \prod_{i=1}^N F_{Z_i}(t) = \prod_{i=1}^N (1 - e^{-p_i t})$  and then

$$\mathbb{E}[Z] = \sum_0^{+\infty} \mathbb{P}(Z > t) dt = \sum_0^{+\infty} \left(1 - \prod_{i=1}^N (1 - e^{-p_i t})\right) dt \tag{A.19}$$

From the above expression, we clearly see that when  $p_i$  is defined by a non-uniform distribution, the number of draws is proportionally larger in  $N$ . However, our original goal is to only sample from the most probably top- $k$  samples, in which case  $\mathbb{E}[Z]$  is lower.

### A.2.7 Additional Results

Figure A.2 is a boxplot of how the performance changes for different KD methods as the student-teacher capacity gap varies. The purpose of this is to identify how much the performance increases are due to larger capacity as opposed to the particular KD method used.

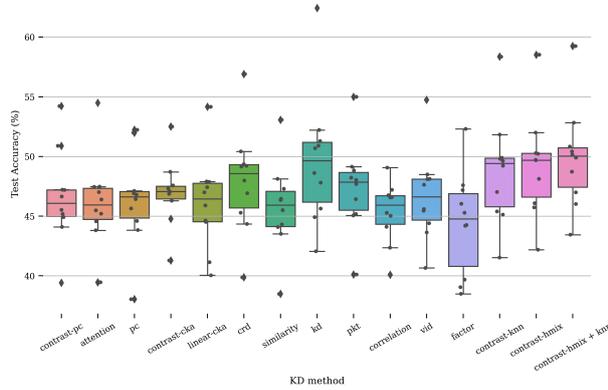


Figure A.2: Tiny-ImageNet Boxplot Test Accuracy for Knowledge Distillation Approaches

Figure A.3 and Figure A.4 (below the references section) show the KD results with the unscaled (i.e no  $[0, 1]$  normalization) color codings and Figure A.5 and Figure A.6 shows the corresponding  $[0, 1]$  row-normalized results to highlight the relative differences between each KD method. We note the last ‘Average Score’ row displays the average

performance over all student-teacher architecture pairs for each KD method.

resnet20-resnet110	69.73	69.69	69.36	70.02	69.70	69.78	71.46	70.48	69.35	69.24	69.91	69.48	68.49	71.66	71.32	72.02
resnet8-resnet110	60.15	59.35	55.69	57.82	58.05	60.48	60.60	59.27	59.47	58.16	60.11	59.93	59.13	61.81	60.91	61.94
resnet14-wrn-16-1	67.36	67.75	63.46	67.10	67.75	67.58	67.75	66.91	66.44	63.85	67.80	68.51	67.63	67.93	68.22	68.21
resnet20-wrn-40-1	69.62	69.62	67.13	69.94	70.60	71.10	71.35	70.13	69.40	68.45	69.58	70.91	69.66	71.97	71.56	72.22
resnet14-wrn-40-1	67.72	67.25	64.80	67.11	67.55	68.87	68.63	66.76	66.38	64.97	67.47	68.44	67.38	71.25	69.62	71.79
resnet32-wrn-16-1	71.22	71.78	67.93	69.03	71.73	72.72	71.56	70.73	71.01	67.21	71.65	72.79	71.50	73.20	72.43	73.93
	Siamese-PC	Linear-CKA	Factor Transfer	Hint	Attention Transfer	Kullback-Leibler Divergence	Contrastive Representational Distillation	Variational Information Distillation	Contrastive-CKA	Similarity Preserving	Correlation Congruence	Probabilistic Knowledge Transfer	Contrastive-PC	InfoNCE-Latent-Mixup	InfoNCE+Class-SCNS	InfoNCE-Latent-Mixup+Class-SCNS

Figure A.3: CIFAR-100 Test Accuracy for KD Approaches

Figure A.7(a), Figure A.7(b), Figure A.7(c), Figure A.7(d), Figure A.7(e) and Figure A.7(f) shows the Tiny-ImageNet-200 embedding similarity of classes and Figure A.7(g), Figure A.7(h), Figure A.7(i), Figure A.7(j) and Figure A.7(k) shows the embedding similarity for CIFAR-100.

### A.3 LAYER FUSION

#### LAYER SIMILARITY VISUALIZATION **Transformers**

##### **Transformer-XL**

We begin by showing the similarity between pretrained layers on Transformer-XL in Figure A.7.

##### **GPT**

##### **GPT-2**

##### **BERT**

**Weight Pruning** We prune a percentage of the weights with the lowest magnitude. This is done in one of two ways: a percentage of weights pruned by layer (layer pruning), or a percentage of the network as whole (global pruning). When used in CNNs, this carried out on convolutional layers.

**Layer Quantization** - We quantize the weights into 128 (i.e 7 bits) clusters post training and decrease the number of clusters as a percentage of 128 (e.g 10% quantized results in 115 clusters). In the subsequent figures that show k-means quantization results, percentages correspond to the number of clusters as a percentage of the number of dimensions in that layer (e.g Quantizing GPT-2 layer  $\theta_l$  at 50% results in 512 clusters since  $|\theta_l| = 1024$ ). Each weight  $\theta_\ell^i$  of the  $i$ -th weight of layer  $\ell$  is assigned to cluster center  $c_\ell \in C$ . For CNNs, all convolutional filters within a layer are flattened  $\mathbb{R}^{c \times f} \rightarrow \mathbb{R}^{cf}$  for  $c$  channels

resnet56-resnet152	45.18	43.81	51.99	47.57	47.87	50.42	48.12	52.22	49.14	42.35	48.49	44.19	52.00	51.83	52.24	52.83	53.00
resnet44-resnet152	47.19	47.45	47.11	48.71	47.90	47.98	47.29	51.29	48.82	45.27	48.12	47.59	50.28	49.89	50.93	50.42	51.00
resnet20-resnet50	45.52	45.48	45.64	46.29	45.89	46.92	44.30	47.80	46.42	46.59	45.59	46.03	48.12	47.02	47.44	48.72	49.00
resnet32-resnet50	46.64	47.47	46.42	46.87	47.42	49.15	45.50	50.69	48.03	46.56	47.63	45.28	50.23	49.80	50.04	50.83	51.00
resnet32-resnet152	50.89	47.01	46.91	47.31	41.13	49.35	46.45	50.90	47.69	46.75	48.09	47.18	49.68	49.62	50.27	50.15	50.00
resnet32-densenet121	47.21	46.39	46.80	47.07	46.99	49.21	46.35	48.61	48.22	47.20	43.63	39.04	49.71	49.22	50.38	49.90	50.00
resnet8-resnet50	39.40	39.45	38.03	41.28	40.03	39.86	38.47	42.05	40.10	40.08	40.65	39.67	42.18	41.52	41.29	43.44	43.00
resnet14-resnet50	44.91	44.56	43.83	47.05	44.63	44.34	43.51	44.91	45.18	45.02	44.40	38.48	46.08	45.39	46.12	46.99	47.00
resnet14-resnet34	44.09	45.20	44.59	44.76	44.51	45.28	44.08	45.63	45.05	44.10	45.45	44.25	45.73	45.12	45.80	46.01	46.00
resnet32x4-resnet50	54.22	54.49	52.23	52.50	54.16	56.91	53.07	62.44	55.00	49.06	54.75	52.31	58.53	58.36	58.95	59.25	60.00
	Siamese-PC	Attention Transfer	PC	Contrastive-CKA	Linear-CKA	Contrastive Representational Distillation	Similarity Preserving	Kullback-Leibler Divergence	Probabilistic Knowledge Transfer	Correlation Congruence	Variational Information Distillation	Factor Transfer	InfoNCE-Latent-Mixup	InfoNCE+Class-SCNS	InfoNCE+Instance-SCNS	InfoNCE+Latent-Mixup+Class-SCNS	InfoNCE+Latent-Mixup+Instance-SCNS

Figure A.4: Tiny-Imagenet 200 Test Accuracy for KD Approaches

resnet20-resnet110	0.35	0.34	0.25	0.43	0.34	0.37	0.84	0.56	0.24	0.21	0.40	0.28	0.00	0.90	0.80	1.00
resnet8-resnet110	0.71	0.59	0.00	0.34	0.38	0.77	0.79	0.57	0.60	0.40	0.71	0.68	0.55	0.98	0.84	1.00
resnet14-wrn-16-1	0.77	0.85	0.00	0.72	0.85	0.82	0.85	0.68	0.59	0.08	0.86	1.00	0.83	0.89	0.94	0.94
resnet20-wrn-40-1	0.49	0.49	0.00	0.55	0.68	0.78	0.83	0.59	0.45	0.26	0.48	0.74	0.50	0.95	0.87	1.00
resnet14-wrn-40-1	0.42	0.35	0.00	0.33	0.39	0.58	0.55	0.28	0.23	0.02	0.38	0.52	0.37	0.92	0.69	1.00
resnet32-wrn-16-1	0.60	0.68	0.11	0.27	0.67	0.82	0.65	0.52	0.57	0.00	0.66	0.83	0.64	0.89	0.78	1.00
Average Score	0.56	0.55	0.06	0.44	0.55	0.69	0.75	0.54	0.45	0.16	0.58	0.68	0.48	0.92	0.82	0.99
	Siamese-PC	Linear-CKA	Factor Transfer	Hint	Attention Transfer	Kullback-Leibler Divergence	Contrastive Representational Distillation	Variational Information Distillation	Contrastive-CKA	Similarity Preserving	Correlation Congruence	Probabilistic Knowledge Transfer	Contrastive-PC	InfoNCE-Latent-Mixup	InfoNCE+Class-SCNS	InfoNCE+Latent-Mixup+Class-SCNS

Figure A.5: CIFAR-100 Normalized Test Accuracy for Knowledge Distillation Approaches

and  $f$  filters and assigned the centers and then reshaped to their original dimensions  $\mathbb{R}^{cf} \rightarrow \mathbb{R}^{c \times f}$  prior to pooling. We minimize the standard k-means objective as  $\arg \min_C \sum_{\ell=1}^L \sum_{i=1}^k \sum_{\theta_\ell^i \in C_\ell^i} |\theta_\ell^i - c_\ell^i|^2$  [259].

**Weight Pruning** We prune a percentage of the weights with the lowest magnitude. This is done in one of two ways: a percentage of weights pruned by layer (layer pruning), or a percentage of the network as whole (global pruning). When used in CNNs, this carried out on convolutional layers.

**Knowledge Distillation** - we follow a slightly different approach than that described by Hinton, Vinyals, and Dean in distilling networks, to deal with retraining such large models. Since it is too expensive to train a smaller student network, we instead iteratively reduce the dimensionality of the existing network during retraining by reconstructing the weights of the original network using denoising autoencoders and truncated SVD for each layer. In the case of autoencoding attention layers, we re-normalize the attention layers using the `softmax` such that each row

resnet56-resnet152	0.27	0.14	0.90	0.49	0.52	0.76	0.54	0.93	0.64	0.00	0.58	0.17	0.91	0.89	0.93	0.98	1.00
resnet44-resnet152	0.32	0.36	0.31	0.57	0.44	0.45	0.34	1.00	0.59	0.00	0.47	0.39	0.83	0.77	0.94	0.86	1.00
resnet20-resnet50	0.26	0.25	0.28	0.42	0.33	0.55	0.00	0.73	0.44	0.48	0.27	0.36	0.80	0.57	0.66	0.93	1.00
resnet32-resnet50	0.23	0.37	0.19	0.27	0.36	0.66	0.04	0.92	0.47	0.22	0.40	0.00	0.84	0.77	0.81	0.94	1.00
resnet32-resnet152	1.00	0.60	0.59	0.63	0.00	0.84	0.54	1.00	0.67	0.58	0.71	0.62	0.88	0.87	0.94	0.92	0.95
resnet32-densenet121	0.72	0.65	0.68	0.71	0.70	0.90	0.64	0.84	0.81	0.72	0.40	0.00	0.94	0.90	1.00	0.96	0.98
resnet8-resnet50	0.23	0.24	0.00	0.55	0.34	0.31	0.08	0.69	0.35	0.35	0.45	0.28	0.71	0.60	0.56	0.92	1.00
resnet14-resnet50	0.73	0.69	0.61	0.97	0.70	0.66	0.57	0.73	0.76	0.74	0.67	0.00	0.86	0.78	0.86	0.96	1.00
resnet14-resnet34	0.00	0.41	0.19	0.25	0.16	0.44	0.00	0.56	0.35	0.01	0.50	0.06	0.60	0.38	0.63	0.70	1.00
resnet32x4-resnet50	0.39	0.41	0.24	0.26	0.38	0.59	0.30	1.00	0.44	0.00	0.43	0.24	0.71	0.70	0.74	0.76	0.82
Average Score	0.41	0.41	0.40	0.51	0.39	0.61	0.30	0.84	0.55	0.31	0.49	0.21	0.81	0.72	0.81	0.89	0.97
	Siamese-PC	Attention Transfer	PC	Contrastive-CKA	Linear-CKA	Contrastive Representati Distillation	Similarity Preserving	Kullback-Leibler Divergence	Probabilistic Knowledge Transfer	Correlation Congruence	Variational Information Distillation	Factor Transfer	InfoNCE-Latent-Mixup	InfoNCE+Class-SCNS	InfoNCE+Instance-SCNS	InfoNCE-Latent-Mixup+Class-SCNS+	InfoNCE-Latent-Mixup+Instance-SCNS

Figure A.6: Tiny-ImageNet 200 Test Accuracy for Knowledge Distillation Approaches

of the resulting  $\mathbb{R}^{n \times n}$  matrix  $\sum_i = 1$  in the self-attention block, where  $n$  is the length of the segment. We also re-normalize attention layers after pruning steps. The cost  $C(z_\ell, \tilde{z}_\ell)$  is then defined as the Mean Squared Error (MSE) as shown in Equation A.20.

$$\tilde{z}_\ell = \text{ReLU}(z_\ell \omega_\ell + \beta_\ell + \epsilon_\ell) \quad \forall \ell \in L, \quad (\text{A.20})$$

$$C(z_\ell, \tilde{z}_\ell) = \arg \min_{\omega} \mathbb{E}_z [\mathcal{L}_2(z_\ell, \tilde{z}_\ell)] \quad (\text{A.21})$$

Similarly we use randomized truncated SVD [139] where  $U \in \mathbb{R}^{m \times m}$  is an orthogonal matrix of left singular vectors,  $\Sigma \in \mathbb{R}^{m \times n}$  diagonal matrix of singular values, and  $V \in \mathbb{R}^{n \times n}$  orthogonal matrix of right singular vectors. This reduces the number of flops of classical SVD computation from  $\mathcal{O}(mnk)$  to  $\mathcal{O}(mn \log(k))$  for  $k$  dominant components using randomized techniques. We can perform QR factorization on  $\theta_\ell$  such that  $Q_\ell^T \theta_\ell = R_\ell$ . Here,  $Q_\ell$  has orthonormal columns of  $\theta_\ell$ . Randomized techniques are used to approximate the range of  $\theta_\ell$  by finding a  $Q_\ell$ . We first find  $B_\ell = Q_\ell^T \theta_\ell$ . Now, as  $B_\ell$  is relatively small (recall  $Q_\ell$  has only a small number of columns), we can efficiently compute the SVD of B by standard methods, and thus  $B_\ell = S_\ell \Sigma_\ell V_\ell^T$  for  $S_\ell, V_\ell$  orthogonal and  $\Sigma_\ell$  diagonal. Then as  $A_\ell \approx Q_\ell Q_\ell^T A_\ell = Q_\ell (S_\ell \Sigma_\ell V_\ell^T)$ , we see that taking  $U_\ell = Q_\ell S_\ell$ , we have computed a low rank approximation  $A_\ell \approx U_\ell \Sigma_\ell V_\ell^T$ .

### A.3.1 Layer Fusion & Compression ReTraining Details

For our proposed LF approach, we consider the aforementioned distances, divergences and alignment measures to choose the top-k layer pairs for LF. For fusing, we consider layer averaging, layer mixing and layer freezing. Most aforementioned prior work that focuses on compression methods with retraining [144] and combinations thereof (e.g [144]

prune, quantize, followed by huffman coding) focus on solely on reducing the network size. In contrast, [111] focused on identifying *lottery tickets* by identifying a subnetwork post-pruning and then retraining from the random initializations of the unpruned weights prior to training of that subnetwork. In contrast, we compress an already pretrained model for both CNNs and Transformers as our aim is to reduce the network while preserving network density and being close to the original performance, as opposed to identifying subnetworks. To maintain uniformity across each compression step, we prune, quantize, fuse and decompose a percentage of the weights for model reduction as opposed to thresholding the weight magnitudes. This ensures a consistent comparison across the compression methods (e.g thresholding weights in pruning does not have a direct equivalent to quantization or weight decomposition, unless we dynamically reduce the network size proportional to the number of weights pruned via thresholding). For retraining we consider two main schemes: (1) for each retraining step we carry out network compression (e.g via pruning) and then retrain remaining network and iteratively carried this out until the final epoch, or (2) in the case where network compression leads to non-zero weights (e.g weight sharing or LF), we freeze the network weights apart from those which have been identified for LF in which case we retrain before tying.

**FUSING LAYERS OF UNEQUAL SIZE** For a pair of vectorized weight tensors of size  $d$  and  $d + k$ , we remove  $k$  weights that have the smallest magnitudes from the  $2^{nd}$  tensor until both match. We also considered using PCA and SVD to fix the dimension for each layer pair. However, this is less of an issue in the case of convolutional layers since filters that are most similar tend to be in the same layer. For transformer models, the same parts of each attention block are fused e.g the key weight tensor from one layer could not be fused with a value weight tensor from another, only another key weight tensor, and these are the same dimensions, hence the same length when vectorized.

### A.3.2 Approximating the Covariance Matrix

For our experiments, some of the layers can be relatively large. For example, the large GPT-2 a weight matrix from a given hidden layer is  $\mathbf{w} \in \mathbb{R}^{2048 \times 2048}$  and a total of 4,194,304 parameters. We split  $\mathbf{W}$  into block matrices to perform covariance estimation. The row  $d_r$  and column  $d + c$  dimensionality are restricted to  $d_r \leq d_r \leq 128$ . A submatrix  $\mathbf{w}_{i,j}$  can be represented as,

$$\mathbf{w}_{i,j} = \begin{bmatrix} \mathbf{w}_{1,1} & \cdots & \mathbf{w}_{1,d_c} \\ \vdots & \ddots & \\ \mathbf{w}_{d_r,1} & & \mathbf{w}_{d_r,d_c} \end{bmatrix}$$

and all submatrices of  $\mathbf{W}$  are then formed as,

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_{1,1} & \mathbf{w}_{1,2} & \cdots & \mathbf{w}_{1,m} \\ \vdots & \ddots & & \\ \vdots & & \ddots & \\ \mathbf{w}_{n,1} & & & \mathbf{w}_{m,n} \end{bmatrix}$$

where  $m = \text{round}(d_r/128)$  and  $n = \text{round}(d_c/128)$ .

Given a pair of submatrices from two adjacent layers  $\mathbf{w}_{i,j}^{n_\ell} \subset \mathbf{W}^{n_\ell}$  and  $\mathbf{w}_{i,j}^{n_{\ell+1}} \subset \mathbf{W}^{n_{\ell+1}}$  from layers  $n_\ell$  and  $n_{\ell+1}$  respectively, we estimate the covariance similarity between. This is computed for all adjacent  $m, n$  submatrix pairs, assuming that  $d_r^{n_\ell} = d_r^{n_{\ell+1}}$  and  $d_c^{n_\ell} = d_c^{n_{\ell+1}}$ . The expected value of the full covariance similarity is then estimated by its mean as,

$$\mathbb{E}[\Sigma_{\mathbf{W}}] = \frac{1}{mn} \sum_{i=1}^m \sum_{j=1}^n d_{cov}(\Sigma_{\mathbf{w}_{i,j}^{n_\ell}}, \Sigma_{\mathbf{w}_{i,j}^{n_{\ell+1}}}) \quad (\text{A.22})$$

The covariance estimation techniques are then applied to compute similarity between covariance matrices.

### A.3.3 Similarity Between Covariance Measures

Here, we detail existing similarity measures for covariance matrices, which in the context of our work are the covariances of the a layers weights in a neural network.

**RIEMMANIAN METRIC** A basic similarity measure between covariances  $\Sigma_{\mathbf{W}_1}, \Sigma_{\mathbf{W}_2}$  could be the vectorized  $\text{vec}(\Sigma_{\mathbf{W}_1}), \text{vec}(\Sigma_{\mathbf{W}_2}) \in \mathbb{R}^{(d^2+d)/2}$  and the  $\ell_1$  or  $\ell_2$  distance can be used for the Euclidean space. However, vectorization destructs the covariance structure of the weights from each neuron (i.e a row vector from  $\Sigma_{\mathbf{W}}$ ). Ideally, we want to preserve which neuron a weight came from and maintain the manifold over the parameter space. Since the parameter covariance is positive semi-definite, it can be viewed as a Riemmanian manifold and we can consider a Riemannian Metric such as the Affine Invariant Riemannian Metric (AIRM) [23, 219, 334] expressed as [23],

$$d_{\text{AIRM}}(\Sigma_{\mathbf{W}_1}, \Sigma_{\mathbf{W}_2}) := \|\log(\Sigma_{\mathbf{W}_1}^{-1/2} \Sigma_{\mathbf{W}_2} \Sigma_{\mathbf{W}_1}^{-1/2})\|_F \quad (\text{A.23})$$

where  $F$  is the Frobenius norm of the resulting matrix. However, this can be slow to compute due to eigenvalue computation and the log of the parameter matrix, which for wide networks is more expensive.

The Log-Euclidean Metric [14, 15] shown in Equation A.24 reduces the computation by converting to symmetric matrices which can then

$$d_{\text{LERM}}(\Sigma_{\mathbf{w}_1}, \Sigma_{\mathbf{w}_2}) := \|\log(\Sigma_{\mathbf{w}_1}) - \log(\Sigma_{\mathbf{w}_2})\|_F \quad (\text{A.24})$$

However, the main drawback with the aforementioned AIRMs is the additional computation involved in preserving the curvature of the space of positive definite matrices. The symmetrized KL-Divergence (KLDM) has also been used by combining the KL divergence in both directions as

$$d_{\text{SKL}}(\Sigma_{\mathbf{w}_1}, \Sigma_{\mathbf{w}_2}) = \frac{1}{2} \left( \text{KL}[\Sigma_{\mathbf{w}_1} \|\Sigma_{\mathbf{w}_2}] + \text{KL}[\Sigma_{\mathbf{w}_2} \|\Sigma_{\mathbf{w}_1}] \right) \quad (\text{A.25})$$

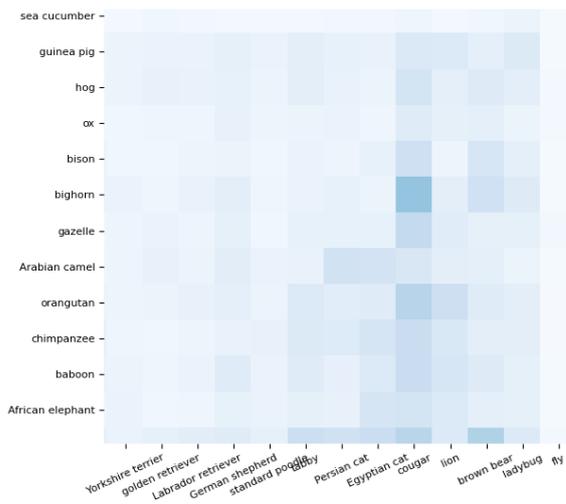
where  $\Sigma_{\mathbf{w}}$  is vectorized. This overcomes the problem of KL not being symmetric and is invariant to inversion so that  $d_{\text{KL}}(\Sigma_{\mathbf{w}_1}^{-1}, \Sigma_{\mathbf{w}_1}^{-1}) = d_{\text{KL}}(\Sigma_{\mathbf{w}_1}, \Sigma_{\mathbf{w}_1})$ .

#### A.3.3.1 Bures-Wasserstein Distance Between Covariances

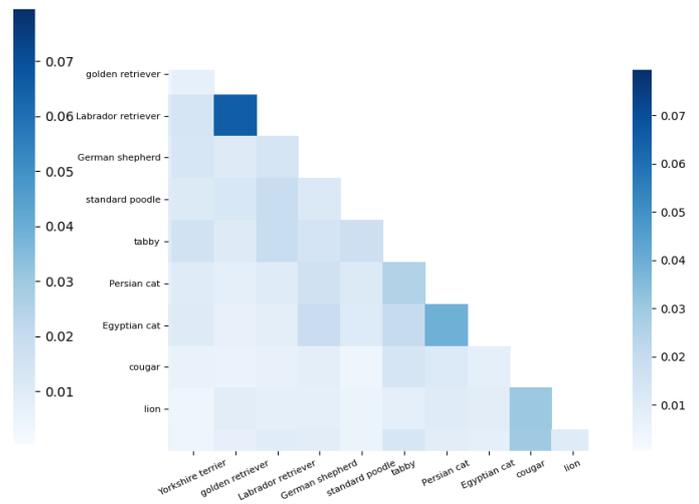
In our work we mainly focused on the Bures-Wasserstein Metric for directly computing the distance between weight matrices. However, an alternative approach is to minimize optimal transport between their covariances [36], which can be expressed as,

$$d_{WS-2}(\Sigma_{\mathbf{w}_1}, \Sigma_{\mathbf{w}_2}) = \left[ \text{tr}\Sigma_{\mathbf{w}_1} + \text{tr}\Sigma_{\mathbf{w}_2} - 2\text{tr}(\Sigma_{\mathbf{w}_1}^{1/2}\Sigma_{\mathbf{w}_2}\Sigma_{\mathbf{w}_1}^{1/2})^{1/2} \right]^{1/2} \quad (\text{A.26})$$

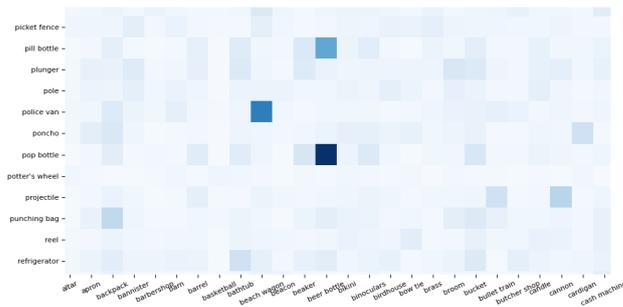
We note that when we only use the variances from the diagonal of  $\Sigma_{\mathbf{w}_1}$ , the  $d_{WS-2}$  becomes the Hellinger distance [155] between the variances of each weight matrix.



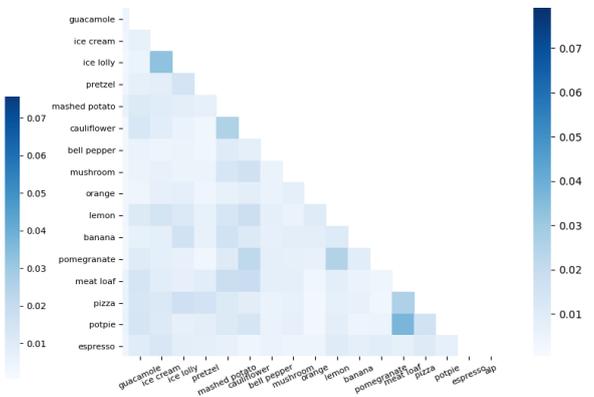
(a) Correlation Plot 1



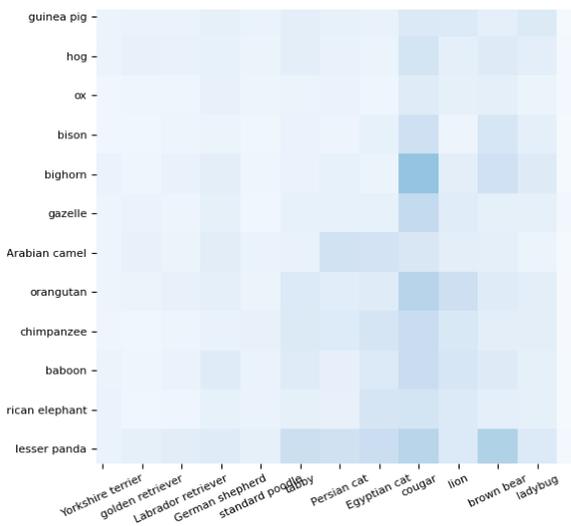
(b) Correlation Plot 2



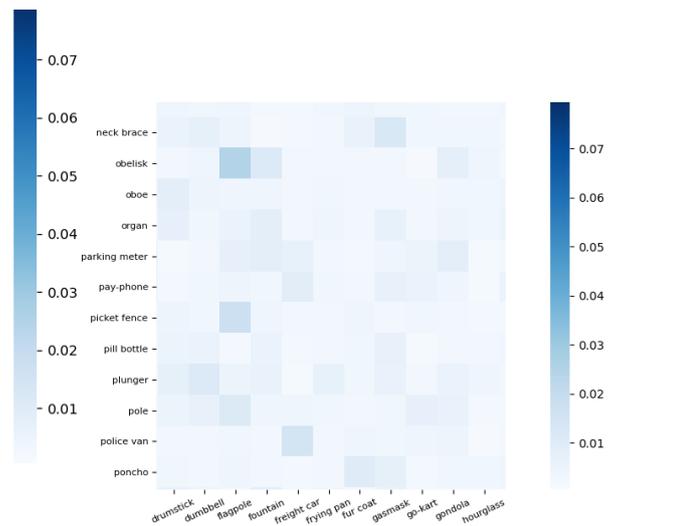
(c) Correlation Plot 3



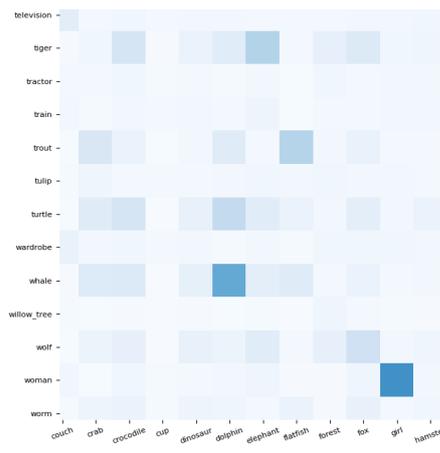
(d) Correlation Plot 4



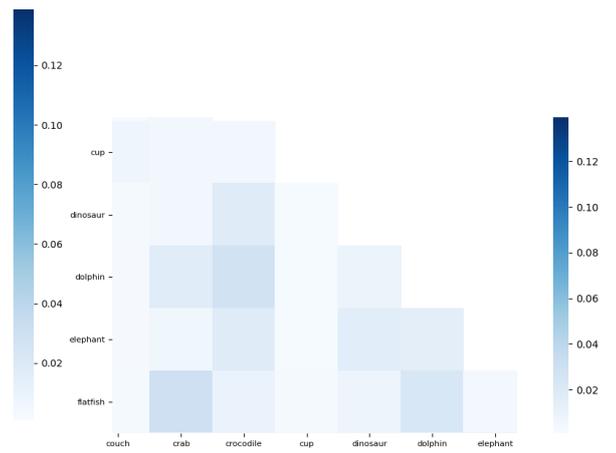
(e) Correlation Plot 5



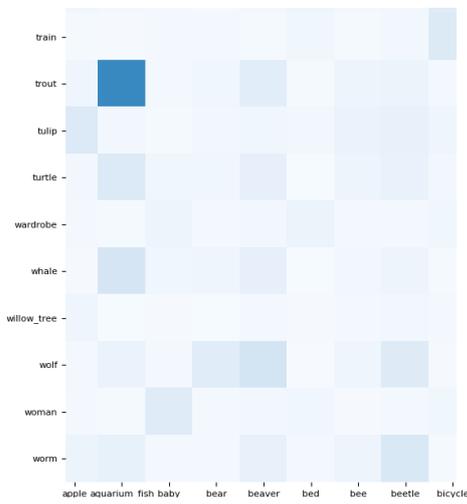
(f) Correlation Plot 6



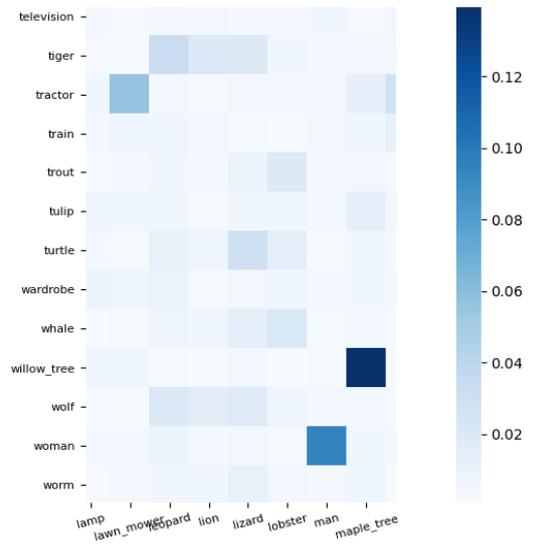
(g) Correlation Plot 1



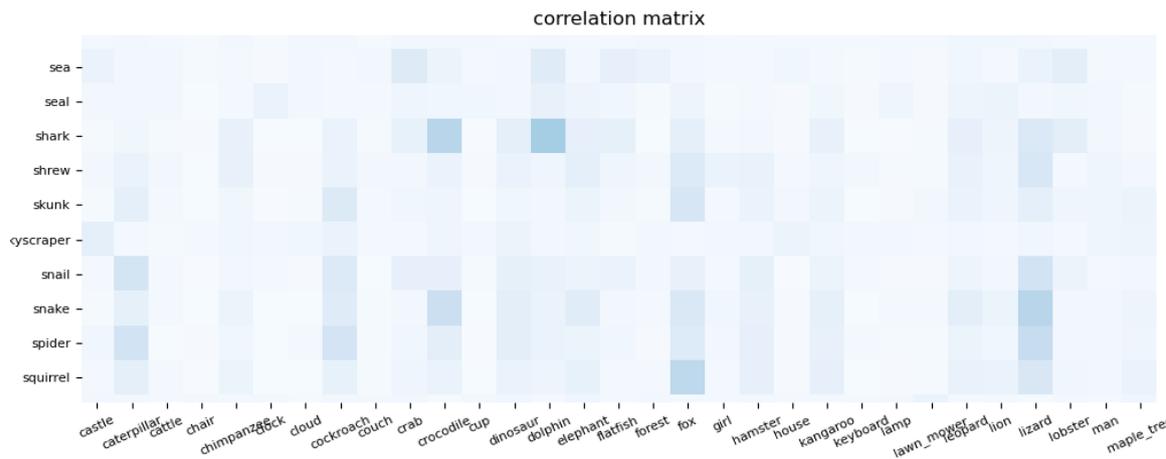
(h) Correlation Plot 2



(i) Correlation Plot 3



(j) Correlation Plot 4



(k) Correlation Plot 4



Figure A.7: Transformer-XL Weight Similarity via Euclidean Distance

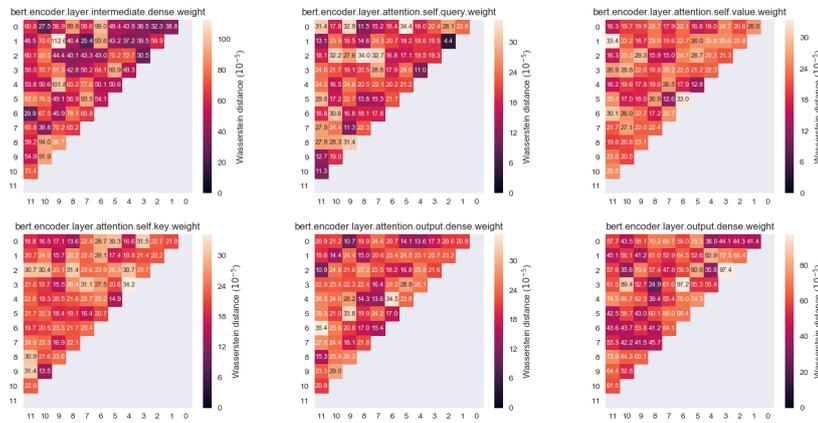


Figure A.8: BERT Weight Similarity with Euclidean Distance

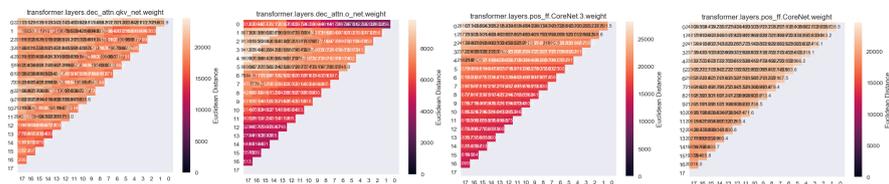


Figure A.9: Euclidean Distance Between Trans-XL Weights: (1) Query-Key-Value Attention, (2) Output Attention, (3, 4) FC Layers