National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics
Speech and Language Processing Group

Structured Pruning for Deep Learning Language Models

Diploma Thesis

of

STEFANOS STAMATIS S. ACHLATIS

Supervisor: Alexandros Potamianos
Associate Professor, NTUA

Athens, November 2021

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics
Speech and Language Processing Group

Structured Pruning for Deep Learning Language Models

DIPLOMA THESIS

of

STEFANOS STAMATIS S. ACHLATIS

Supervisor: Alexandros Potamianos
Associate Professor, NTUA

Approved by the examination committee on 9th November 2021.

(Signature)	*(Signature)*	*(Signature)*
.....	.....	.....
Alexandros Potamianos	Constantinos Tzafestas	Athanasios Katsamanis
Associate Professor, NTUA	Associate Professor, NTUA	Researcher B', RC Athena

Athens, November 2021

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics
Speech and Language Processing Group

(Signature)

.....
Stefanos Stamatis S. Achlatis

Electrical & Computer
Engineer

Περίληψη

Στην παρούσα Διπλωματική Εργασία μελετάμε την συμπίεση Βαθιών Νευρωνικών Δικτύων και πιο συγκεκριμένα την Δομική Περικοπή (Structured Pruning) σε μοντέλα Επεξεργασίας Φυσικής Γλώσσας. Η παρούσα μελέτη μπορεί να προσεγγιστεί από δύο ισοδύναμες οπτικές:

- **Μελέτη της περικοπής για την περικοπή.** Καταλήξαμε στο συμπέρασμα ότι είναι προτιμότερο να εφαρμόζονται στρατηγικές περικοπής που μελετάνε ταυτόχρονα τόσο το προεκπαιδευμένο (pre-trained) όσο και το μετεκπαιδευμένο (fine-tuned) μοντέλο.
- **Μελέτη της περικοπής για την καλύτερη κατανόηση του μοντέλου.** Καταλήξαμε στο συμπέρασμα ότι με την μετεκπαίδευση το μοντέλο τείνει να ξεχνάει την γνώση που είχε μάθει από την προεκπαίδευσης, και για να υπερκεράσουμε αυτό το πρόβλημα κατά την διάρκεια της δομικής περικοπής μελετάμε ταυτόχρονα τόσο το προεκπαιδευμένο όσο και το μετεκπαιδευμένο μοντέλο.

Πολλές μελέτες έχουν γίνει στον τομέα της συμπίεσης νευρωνικών δικτύων και εφαρμόζουν διαφορετικές τεχνικές όπως την Περικοπή Βαρών, την Δομική Περικοπή, την Κβαντοποίηση, την Απόσταξη Γνώσης και την Εύρεσης Νευρωνικής Αρχιτεκτονικής. Ωστόσο, μέσω της Δομικής Περικοπής μπορούμε να συμπιέσουμε το δίκτυο και ταυτόχρονα να καταλάβουμε τα θεμελιώδη δομικά τμήματά του. Για αυτόν τον λόγο, κάνουμε Δομική Περικοπή σε μοντέλα που βασίζονται στο BERT, που είναι το κυρίαρχο μοντέλο στον χώρο της Επεξεργασίας Φυσικής Γλώσσας.

Άλλες μελέτες στον τομέα της δομικής περικοπής του BERT λαμβάνουν υπόψιν μόνο το μετεκπαιδευμένο μοντέλο, παρόλο που το μετεκπαιδευμένο μοντέλο προήλθε από μια διαδικασία κατά την οποία τα βάρη του είναι κυρίως καθορισμένα από την προεκπαίδευση και έχουν λάβει μια μικρή εξειδίκευση για την τελική εργασία. Για αυτό τον λόγο, προτείνουμε μια στρατηγική Δομικής Περικοπής που λαμβάνει υπόψιν τόσο το προεκπαιδευμένο όσο και το μετεκπαιδευμένο μοντέλο και έτσι η σημαντικότητα των κεφαλών υπολογίζεται λαμβάνοντας υπόψιν την σημαντικότητα των αντίστοιχων κεφαλών και στα δύο μοντέλα.

Σε αυτή την εργασία, μελετάμε τον τρόπο με τον οποίο αυτή η ιδέα μπορεί να εφαρμοστεί σε αρχιτεκτονικές τύπου BERT και για να δείξουμε την σημαντικότητα της προσέγγισής μας εκτελούμε πειράματα που λαμβάνουν υπόψιν BERT μοντέλα που είναι προεκπαιδευμένα σε διαφορετικά σύνολα κειμένων και τα ελέγχουμε σε όγκο δεδομένων που αφορούν διαφορετικούς τομείς.

Επίσης, μελετάμε την προσέγγιση μας μέσω της Ὑπόθεσης Τυχερού Δελτίου' (Lottery Ticket Hypothesis) και δείχνουμε ότι για την απόκτηση της αρχικοποίησης μασκών περικοπής είναι καλύτερο να λαμβάνουμε υπόψιν και τα δύο μοντέλα. Ταυτόχρονα, δείχνουμε μια καλύτερη εφαρμογή της "Υπόθεσης Τυχερού Δελτίου" στην Δομική Περικοπή και την ονομάζουμε "Επαναλαμβανόμενη Δομική Περικοπή".

Τέλος, μελετάμε την ικανότητα γενίκευσης της μεθοδολογίας μας σε διαφορετικές τροπικότητες (modalities), όπως στην τροπικότητα της φωνής μέσω του μοντέλου wav2vec 2.0.

Λέξεις Κλειδιά

Βαθιά Μάθηση, Επεξεργασία Φυσικής Γλώσσας, Transfer Learning, Transformers, BERT, Συμπίεση, Δομική Περικοπή, Υπόθεσης Τυχερού Δελτίου, μετεκπαίδευση (fine-tuning)

Abstract

In this Diploma Thesis, we study the compression of Deep Neural Networks, and more precisely, we study the structured pruning in Natural Language Processing models. From our work, we draw some conclusions that can be divided into two main aspects:

- **Studying pruning for pruning.** We propose a better implementation of pruning that considers both the pre-trained and the fine-tuned model.
- **Studying pruning for a better understanding of the model.** We see that through fine-tuning, the model forgets prior knowledge, and in order to overcome this problem, we study both the pre-trained and the fine-tuned model for better pruning results.

Many studies have been done regarding compression neural networks, and they apply different compression techniques such as Magnitude Pruning, Structural Pruning, Quantization, Knowledge Distillation, and Neural Architecture Search. However, through Structured Pruning, one can compress the network and understand the fundamental structured components of the model. Therefore, we focus on Structured Pruning of BERT-based models, the dominant models used in Natural Language Processing.

Other studies regarding Structured Pruning of BERT-based models considered only the final fine-tuned model, even though these models are created after a fine-tuning process, where weight values are mostly predetermined by the original model and are only fine-tuned on the end task. Thus, we suggest that pruning strategies should both consider the pre-trained and the final fine-tuned model, and the head importance score should be calculated considering both the importance of the pre-trained and the fine-tuned head.

In this study, we examine how this idea could be implemented for BERT-base models, and in order to illustrate the impact of our approach, we execute experiments considering BERT models pre-trained on different corpora and fine-tuned on datasets of different domains.

Moreover, we study our approach through the Lottery Ticket Hypothesis, where we see that obtaining initialization pruning masks considering both the pre-trained and the fine-tuned model outperforms the approach which only considers the fine-tuned model. Moreover, we propose a better application of the Lottery Ticket Hypothesis in structured pruning, and we name this approach "Iterative Structural Pruning".

Last but not least, we examine the generalization ability of our methodology through different modalities, and we examine a speech model named wav2vec 2.0. This study is essential because many Transformer-based architectures achieve significant results on different modalities such as speech and vision.

With this thesis, we wish to open new roads and create new aspects to explore pruning mechanisms, Lottery Ticket Hypothesis, and fine-tuning techniques.

Keywords

Deep Learning, Natural Language Processing, Transfer Learning, Transformers, BERT, Compression, Structured Pruning, Lottery Ticket Hypothesis, fine-tuning

To My Mother,
Christina

Ευχαριστίες

Η παρούσα διπλωματική εργασία αποτελεί ένα προσωπικό πόνημα στην διαμόρφωση του οποίου συντέλεσαν πολλοί και διάφοροι άνθρωποι τους οποίους θα ήθελα να ευχαριστήσω.

Αδιαμφισβήτητα, το πρώτο και πιο ουσιαστικό ευχαριστώ οφείλω να το προσφέρω στον επιβλέποντα καθηγητή της παρούσας μελέτης, τον καθηγητή Αλέξανδρο Ποταμιάνο. Η διάθεσή του για βοήθεια, οι συμβουλές του και οι πολυσχιδείς του γνώσεις ήταν καθοριστικές τόσο για την περάτωση αυτού του έργου όσο και για την προσωπική μου πνευματική διαμόρφωση.

Ένα βαθύ ευχαριστώ οφείλω στους διδακτορικούς ερευνητές της ομάδας του κυρίου Ποταμιάνου. Πιο συγκεκριμένα, ευχαριστώ τον Γιώργο Παρασκευόπουλο, τον Ευθύμη Γεωργίου και τον Χάρη Παπαϊωάννου. Οι συμβουλές τους ήταν πολύτιμες και ο ενθουσιασμός τους για την έρευνα αποτελεί προσωπική πηγή έμπνευσης.

Φυσικά, θα ήθελα να ευχαριστήσω όλους τους συμφοιτητές μου και τους φίλους μου, για τις συζητήσεις μας και την ανταλλαγή γνώσεων και απόψεων. Τους ευχαριστώ ειλικρινά.

Τέλος, οφείλω να ευχαριστήσω τους γονείς μου, την Χριστίνα και τον Σωτήρη. Χωρίς την στήριξή τους δεν θα τα είχα καταφέρει.

Στέφανος Σταμάτης Σ. Αχλάτης

Αθήνα, Νοέμβρης 2021

Contents

Περίληψη 1

Abstract 3

Ευχαριστίες 7

List of Figures 13

List of Tables 17

0 Εκτεταμένη Ελληνική Περίληψη 19
- 0.1 Περίληψη . . . 19
- 0.2 Εισαγωγή . . . 19
- 0.3 Σχετική Βιβλιογραφία . . . 20
- 0.4 Διατύπωση Προβλήματος . . . 21
  - 0.4.1 Ορισμός Δομικής Περικοπής . . . 21
  - 0.4.2 BERT Αρχιτεκτονική . . . 21
- 0.5 Προτεινόμενη Μεθοδολογία . . . 21
- 0.6 Πειράματα . . . 22
  - 0.6.1 Διαφορετικά Επίπεδα Περικοπής . . . 22
  - 0.6.2 Ο όρος εξειδίκευσης α . . . 24
  - 0.6.3 Κερδίσαμε το Λαχείο; . . . 27
  - 0.6.4 Επαναλαμβανόμενη Δομική Περικοπή . . . 27
  - 0.6.5 Περικοπή σε Μοντέλα Διαφορετικής Τροπικότητας . . . 28

1 Introduction 29
- 1.1 From Artificial Intelligence To Deep Learning . . . 29
- 1.2 Motivation . . . 29
- 1.3 Research Contributions . . . 30
- 1.4 Thesis Outline . . . 31

2 From Machine Learning to Deep Learning 33
- 2.1 Introduction . . . 33
- 2.2 Learning Classification . . . 34
  - 2.2.1 Supervised Learning . . . 34
  - 2.2.2 Unsupervised Learning . . . 35
  - 2.2.3 Self-Supervised Learning . . . 35
  - 2.2.4 Transfer Learning . . . 35
- 2.3 Machine Learning Concepts . . . 36
  - 2.3.1 Loss Function . . . 36
  - 2.3.2 Underfitting and Overfitting . . . 37

2.3.3 Regularization, Dropout and Pruning . . . 37
2.3.4 Gradient descent . . . 39
2.4 Machine Learning Models . . . 39
2.4.1 Linear Regression . . . 39
2.4.2 Logistic Regression . . . 40
2.4.3 Other Models . . . 41
2.5 Deep Learning Models . . . 42
2.5.1 Feedforward Neural Networks . . . 42
2.5.2 Recurrent Neural Networks . . . 43
2.5.3 Attention Models . . . 45
2.5.4 Transformers . . . 48

3 Natural Language Processing 53
3.1 Introduction . . . 53
3.2 Applications . . . 54
3.3 Word Representation . . . 54
3.3.1 Denotational Representation . . . 54
3.3.2 Distributional Semantics . . . 55
3.4 Language Models . . . 59
3.4.1 Traditional Language Models . . . 59
3.4.2 Neural Language Models . . . 61
3.5 Embeddings from Language Models (ELMo) . . . 62
3.6 Bidirectional Encoder Representations from Transformers (BERT) . . . 62
3.7 GLUE Benchmark . . . 64
3.7.1 CoLA . . . 65
3.7.2 SST-2 . . . 65
3.7.3 MRPC . . . 65
3.7.4 QQP . . . 65
3.7.5 STS-B . . . 66
3.7.6 MNLI . . . 66
3.7.7 QNLI . . . 66
3.7.8 RTE . . . 66
3.7.9 WNLI . . . 66
3.7.10 SciERC . . . 67
3.7.11 PubMed 200k RCT . . . 67

4 Compression of Deep Learning Models 69
4.1 Introduction . . . 69
4.2 Compression: Problem Setting . . . 69
4.2.1 Pruning . . . 70
4.2.2 Quantization . . . 71
4.3 Lottery Ticket Hypothesis (LTH) . . . 72
4.4 Pruning Transformer-based models . . . 73
4.4.1 Transformer-based Structured Pruning . . . 73
4.4.2 Transformer-based Magnitude Pruning . . . 79
4.5 Pruning Computer Vision models . . . 80
4.6 Pruning Using Explainable AI (XAI) Techniques . . . 80

5 Back to the Future: A transfer learning approach for structured pruning 83
5.1 Abstract . . . 83
5.2 Introduction . . . 83
5.3 Related Work . . . 84
5.4 Problem Definition . . . 85
5.4.1 Structured Pruning Definition . . . 85
5.4.2 BERT Architecture . . . 85
5.5 Proposed Method . . . 85
5.6 Experiments . . . 86
5.6.1 Different Pruning Rates . . . 86
5.6.2 The specialization factor α . . . 89
5.6.3 Do we win the Lottery? . . . 91
5.6.4 Iterative Structured Pruning . . . 92
5.6.5 Pruning in a different modality . . . 92

6 Conclusions 95
6.1 Discussion . . . 95
6.2 Future Work . . . 96

Bibliography 97

Bibliography 102

List of Abbreviations 103

List of Figures

1 Αποτελέσματα στα προβλήματα του GLUE. Εκτελούμε πειράματα με διαφορετικές τιμές της υπερπαραμέτρου α και καταγράφουμε τα αποτελέσματα του κομμένου μοντέλου στο σύνολο ανάπτυξης. Κόβουμε 14 κεφαλές σε κάθε βήμα περικοπής. Για $\alpha = 1$ η μεθοδολογία μας είναι ισοδύναμη με την συγκρινόμενη μεθοδολογία του Michel [1]. Για κάθε πείραμα ελέγξαμε 4 διαφορετικά τυχαία φύτρα. 24

2 Λεξιλογική επικάλυψη (%) μεταξύ διαφορετικών τομέων. Ο όρος "PT" δηλώνει ένα δείγμα από πηγή αντίστοιχη της πηγής προεκπαίδευσης του BERT. 25

3 Αποτελέσματα στα σύνολα δεδομένων PubMed 20k RCT και SciERC. Εκτελούμε πειράματα με διαφορετικές τιμές της υπερπαραμέτρου α και καταγράφουμε τα αποτελέσματα του κομμένου μοντέλου στο σύνολο ανάπτυξης. Κόβουμε 14 κεφαλές σε κάθε βήμα περικοπής. Για $\alpha = 1$ η μεθοδολογία μας είναι ισοδύναμη με την συγκρινόμενη μεθοδολογία του Michel [1]. 25

4 Αποτελέσματα στα σύνολα δεδομένων PubMed 20k RCT και SciERC. Εκτελούμε πειράματα με διαφορετικές τιμές της υπερπαραμέτρου α και καταγράφουμε τα αποτελέσματα του κομμένου μοντέλου στο σύνολο ανάπτυξης. Κόβουμε 14 κεφαλές σε κάθε βήμα περικοπής. Για $\alpha = 1$ η μεθοδολογία μας είναι ισοδύναμη με την συγκρινόμενη μεθοδολογία του Michel [1]. 26

5 Επαναλαμβανόμενη Δομική Περικοπή στο MNLI και στο QNLI με $\alpha = 1$ 28

6 Το μοντέλο wav2vec 2.0. έχει μετεκπαιδευτεί στο σύνολο δεδομένων Timit [2] και η επίδοση του καταγράφεται στην μετρική του Word Error Rate. 28

2.1 Artificial Intelligence, Machine Learning and Deep Learning Domains 33

2.2 Training and test errors behave differently. At the left end of the graph, training error and generalization error are both high. This is the underfitting regime. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the overfitting regime, where capacity is too large, above the optimal capacity. Source: [3] 38

2.3 Large learning rate leads the loss function to fluctuates around the minimum. Source: [4] 40

2.4 Loss function's landscapes could be full of local minima, which potentially could lead to a suboptimal solution by GD Source: [4] 40

2.5 Perceptron Structure. Source: cs231n.github.io 42

2.6 Multilayer perceptron (MLP) Structure. Source: electronicdesign 43

2.7 A rolled up RNN where X_t is the input vector containing sequences of characters of a word while h_t is as output vector. Source: colah.github.io 43

2.8 An unrolled up RNN where X_t is the input vector containing sequences of characters of a words while h_t is as output vector. Source: colah.github.io 44

2.9 The repeating module in an LSTM. Source: colah.github.io 45

2.10 Encoder-decoder architecture: (a) traditional (b) with attention model. Source: [5] 47

2.11 Overview of vanilla Transformer architecture. Source: [6] 49
2.12 (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. Source: [7] 50

3.1 The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. Source: [8] 58
3.2 A feed-forward neural network language model. Source: [9] 61
3.3 Pre-training and fine-tuning procedures for BERT. Source: [10] 63
3.4 Task descriptions and statistics. All tasks are single sentence or sentence pair classification, except STS-B, which is a regression task. MNLI has three classes; all other classification tasks have two. Test sets shown in bold use labels that have never been made public in any form. Source: [11] 65

4.1 Graphic Illustration of Lottery Ticket Hypothesis 72
4.2 Early-stopping iteration and accuracy of LeNet under one-shot and iterative pruning. Average of five trials; error bars for the minimum and maximum values. Source: [12] 73
4.3 Typical self-attention classes used for training a neural network. Both axes on every image represent BERT tokens of an input example, and colors denote absolute attention weights (darker colors stand for greater weights). The first three types are most likely associated with language model pre-training, while the last two potentially encode semantic and syntactic information. Source: [13] 74
4.4 FrameNet annotation example for the “address” lexical unit with two core frame elements of different types annotated. Source: [13] 74
4.5 Importance (according to LRP) of self-attention heads. The model trained on 6m OpenSubtitles EN-RU data. Source: [14] 75
4.6 BLEU score as a function of number of re-tained encoder heads (EN-RU). Regularization applied by fine-tuning trained model. Source: [14] 76
4.7 Functions of encoder heads retained after pruning. Each column represents all remaining heads after varying amount of pruning (EN-RU; Subtitles). Source: [14] 76
4.8 The “good” subnetworks for QNLI: self-attention heads (top, 12×12 heatmaps) and MLPs (bottom, 1×12 heatmaps), pruned together. Earlier layers start at 0. The experiment ran with 5 random initializations and reported averages and standard deviations. Source: [15] 78
4.9 The "good" and "bad" subnetworks in BERT fine-tuning: performance on GLUE tasks. ’Pruned’ subnetworks are only pruned, and ’retrained’ subnetworks are restored to pre-trained weights and fine-tuned. Subfigure titles indicate the task and percentage of surviving weights. STD values and error bars indicate standard deviation of surviving weights and performance, respectively, across 5 fine-tuning runs. Source: [15] 78
4.10 Overlaps in BERT’s “good” subnetworks be- tween GLUE tasks: self-attention heads. Source: [15] 79
4.11 Performance of subnetworks at the highest sparsity for which IMP finds winning tickets on each task. To account for fluctuations, we consider a subnetwork to be a winning ticket if its performance is within one standard deviation of the unpruned BERT model. Entries with errors are the average across five runs, and errors are the standard deviations. IMP = iterative magnitude pruning; RP = randomly pruning; θ_0 = the pre-trained weights; θ_0' = random weights; θ_0'' = randomly shuffled pre-trained weights. Source: [16] 80

4.12 Illustration of the evolution of a 5×5 filter with steps of training. Initial training of the network for Task I learns a dense filter as illustrated in (a). After pruning by 60% and re-training, a sparse filter for Task I is obtained, as depicted in (b), where white circles denote 0 valued weights. Weights retained for Task I are kept fixed for the remainder of the method and are not eligible for further pruning. The pruned weights are allowed to be updated for Task II, leading to filter (c), which shares weights learned for Task I. Another round of pruning by 33% and re-training leads to filter (d), the filter used for evaluating Task II (Note that weights for Task I, in gray, are not considered for pruning). Hereafter, weights for Task II, depicted in orange, are kept fixed. This process is completed until desired or runs out of pruned weights, as shown in the filter (e). The final filter (e) for Task III shares weights learned for tasks I and II. At test time, appropriate masks are applied depending on the selected Task to replicate filters learned for the respective tasks. Source: [17] 81

4.13 Overview of Piggyback method fo learning piggyback masks for fixed backbone networks. During training, maintaining a set of real-valued weights m^r which are passed through a thresholding function to obtain binary-valued masks m. These masks are applied to the weights W of the backbone network in an element-wise fashion, keeping individual weights active, or masked out. The gradients obtained through backpropagation of the task-specific loss are used to update the real-valued mask weights. After training, the real-valued mask weights are discarded, and only the thresholded mask is retained, giving one network mask per task. Source: [18] 81

4.14 Illustration of the LRP procedure. Each neuron redistributes to the lower layer as much as it has received from the higher layer. Source: [19] 82

4.15 Input image and pixel-wise explanations of the output neuron 'castle' obtained with various LRP procedures. Parameters are $\varepsilon = 0.25$ std and $\gamma = 0.25$. Source: [19] . 82

5.1 Results on GLUE tasks. We conduct the experiments for different values of the hyperparameter α, and the result is the performance of the model in the development set after pruning 14 attention heads in each pruning step. For $\alpha = 1$ our method is equivalent to the baseline method of Michel et al. [1]. For each experiment, we test 4 random seeds. 88

5.2 Vocabulary overlap (%) between domains. PT denotes a sample from sources similar to BERT pretraining corpus. Vocabularies for each domain are created by considering the top 10K most frequent words (excluding stopwords) in documents sampled from each domain. 89

5.3 Results on PubMed 20k RCT and SciERC on different pruning rates. We conduct the experiments for different values of the hyperparameter α, and the result is the model's performance after pruning 14 attention heads in each pruning step. For $\alpha = 1$ our method is equivalent to the baseline method of Michel et al. [1]. 90

5.4 Results on GLUE tasks and PubMed 20k RCT on different pruning rates. We conduct the experiments for different hyperparameter α values, and the result is the model's performance after pruning 14 attention heads in each pruning step. . 91

5.5 Iterative Structured Pruning (ISP) on MNLI and QNLI with $\alpha = 1$ 93

5.6 wav2vec 2.0 fine-tuned on Timit dataset. The Performance is computed in WER. 93

List of Tables

1 Προβλήματα του GLUE [20], μέγεθος συνόλου δεδομένων, μετρικές αξιολόγησης και υπερπαράμετροι μετεκπαίδευσης που χρησιμοποιήθηκαν στην παρούσα εργασία . . . 23

2 Αποτελέσματα στα προβλήματα του GLUE. Εκτελούμε τα πειράματα με διαφορετικές τιμές της υπερπαραμέτρου α και καταγράφουμε τα αποτελέσματα του κομμένου μοντέλου στο σύνολο ανάπτυξης. Για $\alpha = 1$ η μεθοδολογία μας είναι ισοδύναμη με την συγκρινόμενη μεθοδολογία του Michel [1]. Για κάθε πρόβλημα η καλύτερη επίδοση καταγράφεται με έντονα γράμματα. Για κάθε πείραμα ελέγξαμε 4 διαφορετικά τυχαία φύτρα. 23

3 Αποτελέσματα στα προβλήματα του GLUE. Η πρώτη γραμμή περιγράφει την επίδοση της συγκρινόμενης μεθοδολογίας του Michel [1]. Στις επόμενες γραμμές παρουσιάζουμε την καλύτερη επίδοση του μοντέλου στο σύνολο ελέγχου μετά την περικοπή 98 κεφαλών. Για κάθε πρόβλημα η καλύτερη επίδοση καταγράφεται με έντονα γράμματα. Για κάθε πείραμα ελέγξαμε 4 διαφορετικά τυχαία φύτρα. 24

4 Τα σύνολα δεδομένων PubMed 20k RCT και SciERC, τα μεγέθη των δεδομένων και οι τιμές υπερπαραμέτρων με τις οποίες έγινε η διαδικασία της μετεκπαίδευσης. 25

5 Αποτελέσματα στα προβλήματα του GLUE και στο PubMed 20k RCT. Κόβουμε 112 κεφαλές από τα μοντέλα με διαφορετικές τιμές τις υπερπαραμέτρου α. Μετά ελέγχουμε την επίδοση του μοντέλου και καταγράφουμε το α για το οποίο καταγράψαμε την υψηλότερη επίδοση. 26

6 Αποτελέσματα στα προβλήματα του GLUE. 27

2.1 Encoder-decoder architecture: traditional and with attention model. Notation: $x = (x_1, \ldots, x_T)$: input sequence, T: length of input sequence, h_i: hidden states of encoder, c: context vector, α_{ij}: attention weights over input, s_j: decoder hidden state,y_j: output token, f, g: non-linear functions, a: alignment function, p: distribution function 48

2.2 Summary of Alignment Functions. Notation: $a\,(k_i, q)$: alignment function for query q and key k_i, sim: similarity functions such as cosine, d_k: length of input, $(W, w_{\text{imp}}, W_0, W_1, W_2)$: trainable parameters, b: trainable bias term, act: activation function. 48

2.3 Complexity and parameter counts of self-attention and position-wise FFN 51

2.4 Per-layer complexity, minimum number of sequential operations and maximum path lengths for different layer types. T is the sequence length, D is the representation dimension and K is the kernel size of convolutions 52

5.1 GLUE tasks [20], dataset sizes metrics and fine-tuning hyperparameters reported in this study 87

5.2 Results on GLUE tasks after pruning 112 heads. We conduct the experiments for different values of the hyperparameter α, and the result is the model's performance in the development set after pruning 112 attention heads. For $\alpha = 1$ our method is equivalent to the baseline method of Michel et al. [1]. For each task, the best result is bolded. For each experiment, we test 4 random seeds. 87
5.3 Results on GLUE tasks after pruning 98 heads. The first line describes the performance of the baseline for $\alpha = 1$, and then we present the best performance of the model in the development set after pruning 98 attention heads. For each task, the best result is bolded, and the corresponding best α is mentioned. For each experiment, we test 4 random seeds. 88
5.4 Results on PubMed 20k RCT and SciERC after pruning 112 heads. We conduct the experiments for different values of the hyperparameter α, and the result is the model's performance after pruning 112 attention heads. For $\alpha = 1$ our method is equivalent to the baseline method of Michel et al. [1]. 90
5.5 PubMed 20k RCT and SciERC dataset sizes metrics and fine-tuning hyperparameters reported in this study 90
5.6 Results on GLUE tasks and PubMed 20k RCT after pruning 112 heads. We prune 112 heads from the models with different values of the hyperparameter α. Then we test their performance, and for the best performance, we mention the corresponding α in this Table. 91
5.7 Results on GLUE tasks 92

Chapter 0

Εκτεταμένη Ελληνική Περίληψη

0.1 Περίληψη

Γνωρίζουμε ότι τα Βαθιά Νευρωνικά Δίκτυα μεγάλου μεγέθους δομούνται από μεγάλο πλήθος παραμέτρων πολλών εκ των οποίων δεν συνεισφέρουν και έτσι δύναται να εφαρμοστούν μεθοδολογίες περικοπής τόσο στα βάρη του δικτύου όσο και στις δομικές οντότητες που το αποτελούν. Στον τομέα της Επεξεργασίας Φυσικής Γλώσσας οι αρχιτεκτονικές που έχουν ως βάση τους Transformers έχουν μελετηθεί ως προς την δομική περικοπή των κεφαλών αυτο-προσοχής (self-attention heads) και έχουμε δει αξιόλογα αποτελέσματα. Σε αυτή την μελέτη εξετάζουμε την δομική περικοπή αρχιτεκτονικών που έχουν ως βάση το μοντέλο BERT και θεωρούμε το γεγονός ότι κατά την διαδικασία της μεταφοράς μάθησης (transfer learning) το μοντέλο είναι το αποτέλεσμα μιας διαδικασίας μετεκπαίδευσης (fine-tuning) πάνω σε ένα προεκπαιδευμένο δίκτυο (pre-trained). Με βάση αυτό προτείνουμε μια νέα μεθοδολογία δομικής περικοπής κατά την οποία κόβουμε κεφαλές μελετώντας ταυτόχρονα το προεκπαιδευμένο και το μετεκπαιδευμένο δίκτυο. Επιπλέον, μελετάμε την μεθοδολογία μας υπό το πρίσμα της "Υπόθεσης Τυχερού Δελτίου" (Lottery Ticket Hypothesis) όπου και δείχνουμε ότι η εξαγωγή μασκών περικοπής που προέρχεται από την ταυτόχρονη μελέτη των δύο μοντέλων παράγει καλύτερα αποτελέσματα σε σχέση με την μάσκες περικοπής που παράγονται από την μελέτη μόνο του μοντέλου μετεκπαίδευσης. Σχετικά με την "Υπόθεσης Τυχερού Δελτίου", προτείνουμε μια καλύτερη εφαρμογή της για δομική περικοπή και την ονομάζουμε "Επαναλαμβανόμενη Δομική Περικοπή". Τέλος, εξετάζουμε την τεχνική μας και σε άλλη τροπικότητα (modality) και πιο συγκεκριμένα στην περιοχή της Αυτόματης Αναγνώρισης Φωνής μέσω του μοντέλου wav2vec 2.0 και βρίσκουμε αντίστοιχα αποτελέσματα.

0.2 Εισαγωγή

Πρόσφατα, στον χώρο της Επεξεργασίας Φυσικής Γλώσσας (Natural Language Processing - NLP), έχουμε δει μεγάλη πρόοδο σε πολλές διαφορετικά προβλήματα (tasks), όπως το πρόβλημα της Κατανόησης Φυσικής Γλώσσας (Natural Language Understanding - NLU) και της Εξαγωγής Συμπερασμάτων σε προβλήματα Φυσικής Γλώσσας (Natural Language Inference - NLI). Μεγάλο κομμάτι αυτής της προόδου οφείλεται στην Μεταφορά Μάθησης (Transfer Learning) των αρχιτεκτονικών που έχουν ως βάση τους Transformers, μια αρχιτεκτονική που προτάθηκε από τον Vaswani [7] το 2017.

Αυτά τα μοντέλα αποτελούνται από εκατομμύρια παραμέτρους που κάνουν αργή την αλληλεπίδραση τους και απαιτούν πολλούς ενεργειακούς και υπολογιστικούς πόρους. Μάλιστα, το μέγεθος αυτών των αρχιτεκτονικών φαίνεται να μεγαλώνει όλο και περισσότερο. Πιο συγκεκριμένα, το BERT-base μοντέλο [21] αποτελείται από 110 εκατομμύρια παραμέτρους, το Turing-NLG [22] αποτελείται από 17 δισεκατομμύρια παραμέτρους και το GPT-3 [23] αποτελείται από 175 δισεκατομμύρια.

Ένας τρόπος για να αντιμετωπιστεί το πρόβλημα του μεγάλου μεγέθους αυτών των δικτύων είναι

μέσω της τεχνικής της συμπίεσης (compression) του δικτύου και πιο συγκεκριμένα με την περικοπή βαρών ή δομικών οντοτήτων όπως των κεφαλών αυτο-προσοχής. Πολλοί ερευνητές, όπως η Voita [14], η Kovaleva [13] και ο Michel et al. [1] έχουν δείξει ότι οι αρχιτεκτονικές που έχουν ως βάση τους Transformers έχουν παραμέτρους που μπορούν να περικοπτούν και έχουν καταφέρει να περικόψουν κεφαλές αυτο-προσοχής χωρίς μεγάλη πτώση στην απόδοση του μοντέλου.

Παρόλο που αυτές οι δουλειές έχουν πετύχει αξιοσημείωτα επίπεδα περικοπής των δικτύων, δεν λαμβάνουν υπόψιν ότι το μοντέλο που εξετάζουν έχεθ δημιουργηθεί μέσω μίας διαδικασίας μετεκπαίδευσης (fine-tuning) και γι' αυτό εξαρτάται ιδιαίτερα από το προεκπαιδευμένο δίκτυο. Σε αυτή την εργασία εξετάζουμε μια νέα ευριστική που αναθέτει σε κάθε κεφαλή αυτο-προσοχής μια τιμή μελετώντας ταυτόχρονα και τα δύο μοντέλα. Στην συνέχεια, μελετάμε την μεθοδολογία μας στα προβλήματα του GLUE [11], στο SciERC [24] και στο PubMed 200k RCT [25].

Μελετάμε την μεθοδολογία μας υπό το πρίσμα της "Υπόθεσης Τυχερού Δελτίου" (Lottery Ticket Hypothesis) όπου και δείχνουμε ότι η εξαγωγή μασκών περικοπής που προέρχεται από την ταυτόχρονη μελέτη των δύο μοντέλων παράγει καλύτερα αποτελέσματα σε σχέση με τις μάσκες περικοπής που παράγονται από την αποκλειστική μελέτη του μοντέλου μετεκπαίδευσης. Σχετικά με την "Υπόθεσης Τυχερού Δελτίου", προτείνουμε μια καλύτερη εφαρμογή της για δομική περικοπή και την ονομάζουμε "Επαναλαμβανόμενη Δομική Περικοπή".

Τέλος, εξετάζουμε την τεχνική μας και σε άλλη τροπικότητα (modality) και πιο συγκεκριμένα στην περιοχή της Αυτόματης Αναγνώρισης Φωνής μέσω του μοντέλου wav2vec 2.0 [26] και βρίσκουμε αντίστοιχα αποτελέσματα.

0.3 Σχετική Βιβλιογραφία

Πολλές μελέτες έχουν δείξει ότι το BERT αποτελείται από μεγάλο πλήθος παραμέτρων πολλών εκ των οποίων δεν συνεισφέρουν [27] και έχουν εφαρμόσει διαφορετικές τεχνικές συμπίεσης, όπως η περικοπή βαρών από τον Gordon [28] και την περικοπή δομικών οντοτήτων από την Voita [14], η Kovaleva [13] και ο Michel et al. [1].

Μεθοδολογίες περικοπής που λαμβάνουν υπόψιν τόσο το μετεκπαιδευμένο όσο και το προεκπαιδευμένο μοντέλο δεν είχαν προταθεί στον χώρο της Επεξεργασίας Φυσικής Γλώσσας μέχρι το 2020, όπου ο Sanh [29] πρότεινε μια μεθοδολογία που την ονόμασε "μετακινούμενη περικοπή" ("movement pruning"), η οποία περικόπτει βάρη στο BERT βασισμένη στο πόσο άλλαξαν οι τιμές των βαρών κατά την διαδικασία της μετεκπαίδευσης.

Ο Frank [12] πρότεινε την ιδέα της "Υπόθεσης Τυχερού Δελτίου" (Lottery Ticket Hypothesis) που διατυπώνεται ως εξής: "Ένα νευρωνικό δίκτυο πρόσθιας διάδοσης που είναι πυκνό και οι τιμές των βαρών του είναι τυχαία αρχικοποιημένες περιέχει υποδίκτυα (τυχερά δελτία - winning tickets) που – όταν εκπαιδευτούν απομονωμένα από το υπόλοιπο δίκτυο – πετυχαίνουν όμοια επίδοση στο σύνολο δεδομένων ελέγχου συγκρινόμενη με την επίδοση του αρχικού δικτύου, αν εκπαιδευτούν με αντίστοιχο πλήθος εποχών εκπαίδευσης". Έτσι, ο Frank όχι μόνο πρότεινε μια μεθοδολογία περικοπής αλλά έδειξε και τον πιο βαθύ και ουσιαστικό λόγο για τον οποίο τα βαθιά νευρωνικά δίκτυα δουλεύουν με επιτυχία.

Ο Chen [16] εφάρμοσε την Υπόθεσης Τυχερού Δελτίου περικόπτοντας βάρη στο BERT και πέτυχε αξιοσημείωτα αποτελέσματα δείχνοντας έτσι ότι η Υπόθεσης Τυχερού Δελτίου μπορεί να εφαρμοστεί και σε δίκτυα που δεν είναι τυχαία αρχικοποιημένα αλλά προεκπαιδευμένα. Ο Prasanna [15] εφάρμοσε την ίδια υπόθεση αλλά περικόπτοντας δομικές οντότητες στο BERT και πέτυχε αξιόλογα αποτελέσματα. Πιο συγκεκριμένα, έκοβε κεφαλές που έχουν την μικρότερη επίδοση σύμφωνα με την ευριστική που πρότεινε ο Michel [1].

0.4 Διατύπωση Προβλήματος

0.4.1 Ορισμός Δομικής Περικοπής

Έχοντας ένα σύνολο δεδομένων $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ και ένα επιθυμητό επίπεδο αραιότητας κ, τότε η δομική περικοπή ενός νευρωνικού δικτύου μπορεί να γραφτεί ως το εξής πρόβλημα βελτιστοποίησης με περιορισμούς:

$$\min_{\mathbf{w_s}} L(\mathbf{w_s}; \mathcal{D}) = \min_{\mathbf{w_s}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(\mathbf{w_s}; (\mathbf{x}_i, \mathbf{y}_i)\right)$$
$$\text{με:} \quad \mathbf{w_s} \in \mathbb{R}^m, \quad \|\mathbf{w_s}\|_0 \leq \kappa$$

όπου $\ell(\cdot)$ είναι η συνάρτηση σφάλματος (loss function) , $\mathbf{w_s}$ είναι το σύνολο δομικών παραμέτρων του νευρωνικού δικτύου π.χ. κεφαλές αυτο-προσοχής, m είναι το ολικό πλήθος των δομικών συνόλων και $\|\cdot\|_0$ είναι η τυπική νόρμα L_0.

Δεν υπάρχει αποδοτικός αλγόριθμος για την ελαχιστοποίηση της L_0 νόρμας αφού είναι μη-κυρτή και NP-δύσκολη και απαιτεί συνδυαστική αναζήτηση. Έτσι, το πρόβλημα της δομικής περικοπής είναι NP-δύσκολο πρόβλημα.

0.4.2 BERT Αρχιτεκτονική

Το BERT είναι επι της ουσίας μια στοίβα από επίπεδα κωδικοποίησης (encoder layers) της αρχιτεκτονικής των Transformers Vaswani et al. [7]. Όλα τα επίπεδα έχουν ακριβώς την ίδια δομή: μια πολυ-κεφαλή αυτό-προσοχής (MHAtt - multi-head self-attention) ακολουθούμενη από ένα MLP επίπεδο και απομένων (residual) συνδέσεις μεταξύ τους.

Το MHAtt αποτελείται από N_h ανεξάρτητες παραμετροποιημένες κεφαλές . Μια κεφαλή αυτο-προσοχής h στο επίπεδο l έχει παραμετροποιηθεί με τις εξείς μεταβλητές: $W_k^h, W_q^h, W_v^h \in \mathbb{R}^{d_h \times d}, W_o^h \in \mathbb{R}^{d \times d_h}$ και το d_h συνήθως είναι ίσο με το d/N_h. Δίνοντας n d-διάστατα διανύσματα εισόδου $\mathbf{x} = x_1, x_2, \ldots x_n \in \mathbb{R}^d$, το MHAtt είναι το άθροισμα των εξόδων κάθε ανεξάρτητης κεφαλές εφαρμόζοντας το διάνυσμα εισόδου x :

$$\text{MHAtt}(\boldsymbol{x}, q) = \sum_{h=1}^{N_h} \xi_h \text{Att}_{W_k^h, W_q^h, W_v^h, W_o^h}(\boldsymbol{x}, q)$$

Για να επιτρέψουμε τις διαφορετικές κεφαλές να αλληλοεπιδράσουν μεταξύ τους, οι αρχιτεκτονικές των Transformers εφαρμόζουν σε κάθε επίπεδο ένα μη γραμμικό νευρωνικό δίκτυο πρόσθιας διάδοσης πάνω στην έξοδο του MHAtt [30].

0.5 Προτεινόμενη Μεθοδολογία

Παρόλο που η περικοπή είναι ιδιαίτερα αποδοτική για τα μοντέλα που έχουν προκύψει από επιβλεπόμενη μάθηση, είναι λιγότερο χρήσιμη για τα μοντέλα που έχουν προκύψει από μεταφορά μάθησης. Στην επιβλεπόμενη μάθηση, οι τιμές των βαρών ορίζονται κυρίως από την διαδικασία εκπαίδευσης για το πρόβλημα που καλείται να αντιμετωπίσει το δίκτυο. Ωστόσο, στην μεταφορά μάθησης τα βάρη καθορίζονται κυρίως από το αρχικό προεκπαιδευμένο μοντέλο και υπάρχει μια μικρή αλλαγή των τιμών του κατά την σύντομη διαδικασία της μετεκπαίδευσης. Σε αυτή την εργασία, προτείνουμε μια διαδικασία περικοπής που λαμβάνει υπόψη τόσο το προεκπαιδευμένο όσο και το μετεκπαιδευμένο μοντέλο.

Αρχικά, όπως ο Michel πρότεινε, εισάγουμε τις μεταβλητές μάσκας ξ_h με τιμές στο $\{0, 1\}$, όπου $\xi_h = 1$ δηλώνει ότι η αντίστοιχη κεφαλή h δεν έχει μασκαριστεί, ενώ $\xi_h = 0$ δηλώνει ότι η κεφαλή h

έχει μασκαριστεί. Έτσι, οδηγούμαστε σε μια τροποποίηση του ορισμού του MHAtt ως εξής:

$$\text{MHAtt}(\boldsymbol{x}, q) = \sum_{h=1}^{N_h} \xi_h \text{Att}_{W_k^h, W_q^h, W_v^h, W_o^h}(\boldsymbol{x}, q)$$

Ορίζουμε την εκτιμώμενη απόλυτη ευαισθησία του μοντέλου ως σημαντικότητα για μια τις μεταβλητές μάσκας ξ_h ως:

$$I_h = \mathbb{E}_{x \sim X} \left| \frac{\partial(\alpha * L_1(x) + (1-\alpha) * L_2(x))}{\partial \xi_h} \right|$$

όπου X είναι η κατανομή δεδομένων, $L_1(x)$ το σφάλμα (loss) του μετεκπαιδευμένου μοντέλου στο δείγμα x, $L_2(x)$ το σφάλμα (loss) του μετεκπαιδευμένου μοντέλου στο δείγμα x και $\alpha \in [0, 1]$ είναι ένας όρος εξειδίκευσης, ο οποίος ελέγχει το βάρος του κάθε σφάλματος. Πιο συγκεκριμένα, το $\alpha > 0.5$ δηλώνει ότι η διαδικασία περικοπής θα δώσει μεγαλύτερη προσοχή στο μετεκπαιδευμένο μοντέλο, ενώ το $\alpha < 0.5$ δηλώνει το αντίθετο και το $\alpha = 0.5$ δηλώνει πως ο αλγόριθμος περικοπής θα δώσει εξίσου προσοχή και στα δύο μοντέλα. Η εφαρμογή του αλγόριθμου περικοπής με υπολογισμό της σημαντικότητας των κεφαλών περιγράφεται από τον Αλγόριθμο 0.1.

Algorithm 0.1: *Δομική Περικοπή με υπολογισμό σημαντικότητας*

1: Μετεκπαίδευση του προεκπαιδευμένου μοντέλου
2: Αρχικοποίηση συνόλου μασκών περικοπής των κεφαλών αυτο-προσοχής σε $s = 1^d$ ▷ d: διάσταση
3: **repeat**
4: Υπολογισμός I_h για τις κεφαλές που δεν έχουν περικοπεί
5: Ταξινόμηση των κεφαλών σε φθίνουσα σειρά με βάση το I_h
6: Περικοπή $\kappa\%$ των αρχικών κεφαλών και ενημέρωση του s αντίστοιχα ▷ κ: Υπερπαράμετρος
7: **until** η αραιότητα του s γίνει ίση με s_T ▷ s_T: Όριο Αραιότητας
8: **return** s

Μετά την εξαγωγή των μασκών s, μπορούμε να κάνουμε μια εφάπαξ εφαρμογή της "Υπόθεσης Τυχερού Δελτίου" για τις κεφαλές αυτο-προσοχής: χρησιμοποιούμε τις μάσκες s στο προεκπαιδευμένο BERT και κάνουμε μετεκπαίδευση του νέου δικτύου στο δεδομένο πρόβλημα. Εναλλακτικά, μπορούμε να εφαρμόσουμε την προτεινόμενη τεχνική με όνομα "Επαναλαμβανόμενη Δομική Περικοπή". Η "Επαναλαμβανόμενη Δομική Περικοπή" περιγράφεται από τον Αλγόριθμο 0.2.

Algorithm 0.2: *Επαναλαμβανόμενη Δομική Περικοπή για προεκπαιδευμένα μοντέλα*

1: Μετεκπαίδευση του προεκπαιδευμένου μοντέλου
2: Αρχικοποίηση συνόλου μασκών περικοπής των κεφαλών αυτο-προσοχής σε $s = 1^d$ ▷ d: διάσταση
3: **repeat**
4: Υπολογισμός I_h για τις κεφαλές που δεν έχουν περικοπεί
5: Ταξινόμηση των κεφαλών σε φθίνουσα σειρά με βάση το I_h
6: Στο προεκπαιδευμένο δίκτυο περικοπή $\kappa\%$ των αρχικών κεφαλών και ενημέρωση του s
7: Μετεκπαίδευση του προεκπαιδευμένου μοντέλου
8: **until** η αραιότητα του s γίνει ίση με s_T ▷ s_T: Όριο Αραιότητας
9: **return** s

0.6 Πειράματα

0.6.1 Διαφορετικά Επίπεδα Περικοπής

Όλα τα πειράματα σε αυτή την ενότητα χρησιμοποιούν ως προεκπαιδευμένο μοντέλο το "bert-base-uncased" από την βιβλιοθήκη των Transformers [31]. Το μοντέλο έχει μετεκπαιδευτεί σε 6

Σύνολο Δεδομένων	MNLI	QQP	QNLI	MRPC	SST-2	CoLA
Παραδείγματα Εκπ.	392,704	363,872	104,768	3,680	67,360	8,576
Επαναλ./Εποχές	12,272	11,371	3,274	115	2,105	268
Εποχές	3	3	3	3	3	3
Μέγεθος Batch	32	32	32	32	32	32
Ρυθμός Μάθησης	2×10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-5}
Βελτιστοποιητής	AdamW with $\epsilon = 1 \times 10^{-8}$					
Μετρική Αξιολ.	Matched Acc.	Acc.	Acc.	Acc.	Acc.	Matthew's

Table 1. *Προβλήματα του GLUE [20], μέγεθος συνόλου δεδομένων, μετρικές αξιολόγησης και υπερπαράμετροι μετεκπαίδευσης που χρησιμοποιήθηκαν στην παρούσα εργασία*

α	MNLI	QNLI	QQP	SST-2	MRPC	CoLA
1	0.586	0.564	0.725	0.837	0.484	0.183
0.7	0.589	**0.670**	0.719	0.783	0.589	0.202
0.6	0.586	0.603	**0.750**	0.798	**0.652**	0.206
0.5	0.566	0.594	0.694	**0.843**	0.537	**0.287**
0.4	**0.623**	0.599	0.709	0.832	0.571	0.229

Table 2. *Αποτελέσματα στα προβλήματα του GLUE. Εκτελούμε τα πειράματα με διαφορετικές τιμές της υπερπαραμέτρου α και καταγράφουμε τα αποτελέσματα του κομμένου μοντέλου στο σύνολο ανάπτυξης. Για $\alpha = 1$ η μεθοδολογία μας είναι ισοδύναμη με την συγκρινόμενη μεθοδολογία του Michel [1]. Για κάθε πρόβλημα η καλύτερη επίδοση καταγράφεται με έντονα γράμματα. Για κάθε πείραμα ελέγξαμε 4 διαφορετικά τυχαία φύτρα.*

διαφορετικά προβλήματα του GLUE: MNLI, QQP, QNLI, MRPC, SST-2 και CoLA με την μετρικές που φαίνονται στον Πίνακα 1. Εφαρμόζουμε τον προτεινόμενο αλγόριθμο 0.1 με $\kappa = 10\%$ και ελέγχουμε την επίδοση του μοντέλου στο σύνολο ανάπτυξης (development set), καθώς στο GLUE δεν είναι δημόσια διαθέσιμο το σύνολο ελέγχου (test set).

Βαθιά Επίπεδα Περικοπής. Όπως έχουμε δει από προηγούμενες μελέτες [15] [14] το BERT επιδέχεται βαθιά περικοπή και γι' αυτό σε αυτό το πείραμα εκτελούμε βαθιά περικοπή. Εκτελούμε 8 επαναλήψεις του προτεινόμενου Αλγορίθμου και σε κάθε επανάληψη περικόπτεται 14 κεφαλές, επομένως κόβουμε 112 από τις 144 κεφαλές, που είναι περίπου το 80% των κεφαλών αυτο-προσοχής του BERT. Τα αποτελέσματα αυτού του πειράματος μπορούν να φανούν στον Πίνακα 2.

Από αυτά τα πειράματα μπορούμε να δούμε ότι για βαθιά επίπεδα περικοπής η μεθοδολογία μας είναι καλύτερη από την συγκρινόμενη μεθοδολογία του Michel [1] για κάθε πρόβλημα του GLUE. Έτσι, μπορούμε να παρατηρήσουμε ότι η χρήση του προεκπαιδευμένου και του μετεκπαιδευμένου δικτύου βοηθάει την διαδικασία της περικοπής. Η καλύτερη τιμή της υπερπαραμέτρου α εξαρτάται από την φύση του προβλήματος, κάτι που θα μελετηθεί εκτενώς σε επόμενα πειράματα.

Μικρότερα Επίπεδα Περικοπής. Σε αυτή την ενότητα εκτελούμε πειράματα σε μικρότερα επίπεδα περικοπής, έτσι εκτελούμε 7 επαναλήψεις του προτεινόμενου αλγορίθμου και για κάθε επανάληψη κόβουμε 14 κεφαλές, έτσι συνολικά κόβουμε 98 από τις 144 κεφαλές, που είναι περίπου το 80% των κεφαλών αυτο-προσοχής του BERT. Τα αποτελέσματα αυτού του πειράματος μπορούν να φανούν στον Πίνακα 3.

Από αυτά τα πειράματα μπορούμε να δούμε ότι και για μικρότερα επίπεδα περικοπής η μεθοδολογία μας είναι καλύτερη από την συγκρινόμενη μεθοδολογία του Michel [1] για κάθε πρόβλημα του GLUE. Έτσι, μπορούμε να παρατηρήσουμε ότι η χρήση του προεκπαιδευμένου και του μετεκπαιδευμένου δικτύου βοηθάει την διαδικασία της περικοπής ακόμη και για μικρότερα επίπεδα περικοπής.

Διαφορετικά Επίπεδα Περικοπής. Σε αυτή την ενότητα εκτελούμε πειράματα σε διάφορα επίπεδα περικοπής, έτσι παρουσιάζουμε τα αποτελέσματα του τελικού μοντέλου για κάθε βήμα περικοπής που είναι ίσο με 14 κεφαλές περικοπής. Συνολικά, εκτελούμε 8 βήματα περικοπής και τα αποτελέσματα είναι η μέση τιμή 4 διαφορετικών πειραμάτων με διαφορετικά τυχαία φύτρα. Τα

MNLI	QNLI	QQP	SST-2	MRPC	CoLA
0.717($\alpha = 1$)	0.734	0.791	0.875	0.639	0.286
0.732 ($\alpha = 0.4$)	**0.787** (0.7)	**0.811** (0.6)	**0.881** (0.4)	**0.730** (0.6)	**0.0.394** (0.6)

Table 3. *Αποτελέσματα στα προβλήματα του GLUE. Η πρώτη γραμμή περιγράφει την επίδοση της συγκρινόμενης μεθοδολογίας του Michel [1]. Στις επόμενες γραμμές παρουσιάζουμε την καλύτερη επίδοση του μοντέλου στο σύνολο ελέγχου μετά την περικοπή 98 κεφαλών. Για κάθε πρόβλημα η καλύτερη επίδοση καταγράφεται με έντονα γράμματα. Για κάθε πείραμα ελέγξαμε 4 διαφορετικά τυχαία φύτρα.*

αποτελέσματα αυτού του πειράματος μπορούν να φανούν στην Εικόνα 1.

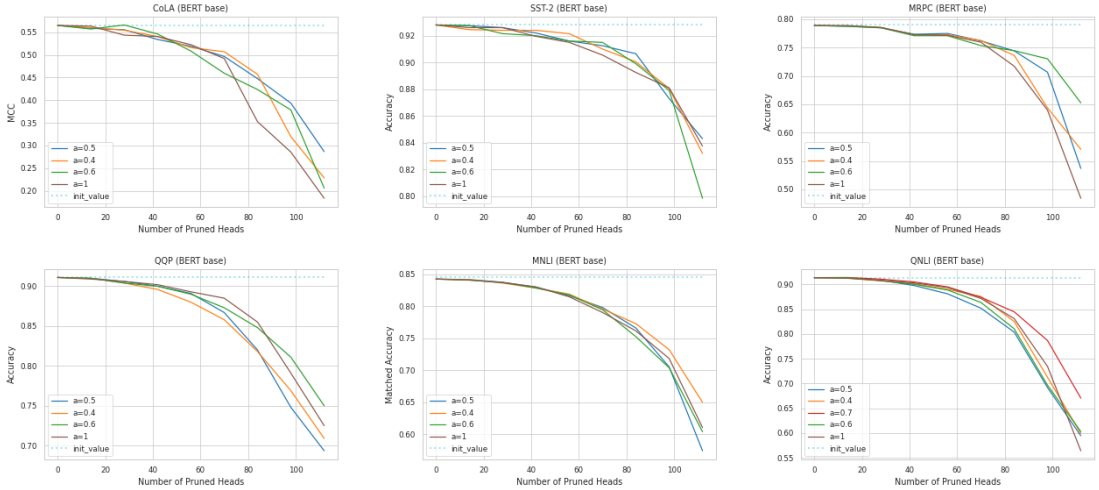


Figure 1. *Αποτελέσματα στα προβλήματα του GLUE. Εκτελούμε πειράματα με διαφορετικές τιμές της υπερπαραμέτρου α και καταγράφουμε τα αποτελέσματα του κομμένου μοντέλου στο σύνολο ανάπτυξης. Κόβουμε 14 κεφαλές σε κάθε βήμα περικοπής. Για $\alpha = 1$ η μεθοδολογία μας είναι ισοδύναμη με την συγκρινόμενη μεθοδολογία του Michel [1]. Για κάθε πείραμα ελέγξαμε 4 διαφορετικά τυχαία φύτρα.*

Από την Εικόνα 1 βλέπουμε ότι η προτεινόμενη μεθοδολογία εν γένει κερδίζει την συγκρινόμενη μεθοδολογία. Παρόλο, που για μικρότερα επίπεδα περικοπής βλέπουμε μικρότερη διαφορά στην επίδοση των διαφορετικών μεθοδολογιών. Αυτό συμβαίνει καθώς το BERT έχει πολλές κεφαλές που περικόπτονται χωρίς μεγάλη επίπτωση στην επίδοση του μοντέλου.

0.6.2 Ο όρος εξειδίκευσης α

Θεωρούμε ότι το α κατά την διάρκεια της διαδικασίας περικοπής ελέγχει το πόσο βάρος δίνουμε στο μετεκπαιδευμένο μοντέλο. Έτσι, το $\alpha > 0.5$ δηλώνει ότι η διαδικασία περικοπής θα δώσει μεγαλύτερη προσοχή στο μετεκπαιδευμένο μοντέλο, ενώ το $\alpha < 0.5$ δηλώνει το αντίθετο και το $\alpha = 0.5$ δηλώνει πως ο αλγόριθμος περικοπής θα δώσει εξίσου προσοχή και στα δύο μοντέλα.

Η προσέγγισή μας θεωρεί ότι το μετεκπαιδευμένο μοντέλο ξεχνάει πολύτιμη πληροφορία από το προεκπαιδευμένο μοντέλο πράγμα που αντιμετωπίζουμε κατά την διάρκεια της περικοπής λαμβάνοντας υπόψη τόσο το μετεκπαιδευμένο όσο και το προεκπαιδευμένο μοντέλο. Επιπλέον, για μικρότερα επίπεδα περικοπής δεν υπάρχουν μεγαλύτερες διαφοροποιήσεις στα αποτελέσματα, ενώ σε μεγαλύτερα επίπεδα ο όρος εξειδίκευσης βοηθάει την διαδικασία.

Για να ελέγξουμε την επίδραση του όρου εξειδίκευσης εξετάζουμε BERT-base μοντέλα σχετικά με Επιστήμη Υπολογιστών και Βιοϊατρική Επιστήμη. Αυτό το πράττουμε καθώς το BERT-base έχει μικρή λεξιλογική επικάλυψη μεταξύ επιστημονικών πεδίων, όπως έχει μελετήσει ο Gururangan [32]

Σύνολο Δεδομένων	PubMed 20k RCT	SciERC
Παραδείγματα Εκπ.	20,000	3,200
Επαναλ./Εποχές	625	100
Εποχές	3	3
Μέγεθος Batch	32	32
Ρυθμός Μάθησης	2×10^{-5}	2×10^{-5}
Βελτιστοποιητής	AdamW with $\epsilon = 1 \times 10^{-8}$	
Μετρική Αξιολ.	Accuracy	Accuracy

Table 4. *Τα σύνολα δεδομένων PubMed 20k RCT και SciERC, τα μεγέθη των δεδομένων και οι τιμές υπερπαραμέτρων με τις οποίες έγινε η διαδικασία της μετεκπαίδευσης.*

και φαίνεται στην Εικόνα 2.

Figure 2. *Λεξιλογική επικάλυψη (%) μεταξύ διαφορετικών τομέων. Ο όρος "PT" δηλώνει ένα δείγμα από πηγή αντίστοιχη της πηγής προεκπαίδευσης του BERT.*

Μοντέλα Μετεκπαιδευμένα σε Επιστημονικά Πεδία . Όλα τα πειράματα σε αυτή την ενότητα έγιναν στο "bert-base-uncased" μοντέλο της βιβλιοθήκης Transformers [31]. Το μοντέλο μετεκπαιδεύτηκε σε 2 επιστημονικά σύνολα δεδομένων: στο SciERC [24] και στο PubMed 20k RCT [25] με τις μετρικές που φαίνονται στον Πίνακα 4. Μετά, εφαρμόστηκε ο προτεινόμενος Αλγόριθμος 5.1 με $\kappa = 10\%$.

Εκτελέσαμε διαδικασία περικοπής για διαφορετικά επίπεδα περικοπής και παρουσιάζουμε τα αποτελέσματα του τελικού μοντέλου με βήμα περικοπής 14 κεφαλές. Συνολικά, εκτελέσαμε 8 βήματα περικοπής. Τα αποτελέσματα περιγράφονται στην Εικόνα 3.

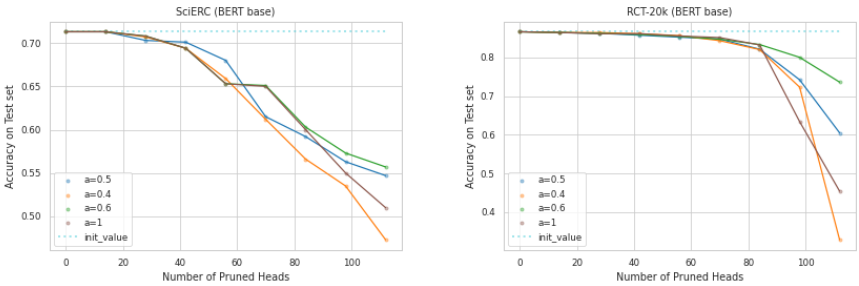


Figure 3. *Αποτελέσματα στα σύνολα δεδομένων PubMed 20k RCT και SciERC. Εκτελούμε πειράματα με διαφορετικές τιμές της υπερπαραμέτρου α και καταγράφουμε τα αποτελέσματα του κομμένου μοντέλου στο σύνολο ανάπτυξης. Κόβουμε 14 κεφαλές σε κάθε βήμα περικοπής. Για $\alpha = 1$ η μεθοδολογία μας είναι ισοδύναμη με την συγκρινόμενη μεθοδολογία του Michel [1].*

Από την Εικόνα 3 βλέπουμε ότι η καλύτερη επίδοση και στα δύο επιστημονικά πεδία πετυχαίνετε

	MNLI	QNLI	SST-2	MRPC	CoLA	PubMed 20k RCT
καλύτερο-α	0.7	0.6	0.6	0.7	1	0.6

Table 5. *Αποτελέσματα στα προβλήματα του GLUE και στο PubMed 20k RCT. Κόβουμε* 112 *κεφαλές από τα μοντέλα με διαφορετικές τιμές τις υπερπαραμέτρου* α*. Μετά ελέγχουμε την επίδοση του μοντέλου και καταγράφουμε το* α *για το οποίο καταγράψαμε την υψηλότερη επίδοση.*

για $\alpha = 0.6$. Αυτό μπορεί να δικαιολογηθεί από το γεγονός ότι το προεκπαιδευμένο BERT έχει εκπαιδευτεί σε κείμενα με μικρή λεξιλογική επικάλυψη με τα επιστημονικά πεδία των σύνολο δεδομένων PubMed 20k RCT και SciERC. Έτσι, η διαδικασία της περικοπής πρέπει να δίνει μεγαλύτερη προσοχή στο μετεκπαιδευμένο μοντέλο.

Μοντέλα με Επιστημονικά Πεδία Προεκπαίδευσης. Σε αυτή την ενότητα, αλλάζουμε το προεκπαιδευμένο μοντέλο και χρησιμοποιούμε το SciBERT [33] το οποίο είναι ένα μοντέλο BERT που έχει λάβει περισσότερη εκπαίδευση σε επιστημονικά κείμενα. Το SciBERT συγκρινόμενο με το BERT έχει μεγαλύτερη λεξιλογική επικάλυψη μεταξύ επιστημονικών πεδίων, όπως η Επιστήμη Υπολογιστών και η Βιοϊατρική Επιστήμη.

Όλα τα πειράματα σε αυτή την ενότητα χρησιμοποιούν ως προεκπαιδευμένο μοντέλο το "bert-base-uncased" από την βιβλιοθήκη των Transformers [31]. Το μοντέλο έχει μετεκπαιδευτεί σε 6 διαφορετικά προβλήματα του GLUE: MNLI, QQP, QNLI, MRPC, SST-2 και CoLA με τις μετρικές που φαίνονται στον Πίνακα 1. Εφαρμόζουμε τον προτεινόμενο αλγόριθμο 0.1 με $\kappa = 10\%$ και ελέγχουμε την επίδοση του μοντέλου στο σύνολο ανάπτυξης (development set), καθώς στο GLUE δεν είναι δημόσια διαθέσιμο το σύνολο ελέγχου (test set).

Το μοντέλο έχει μετεκπαιδευτεί σε 5 διαφορετικά προβλήματα του GLUE: MNLI, QNLI, MRPC, SST-2 and CoLA και σε ένα Επιστημονικό Σύνολο Δεδομένων: PubMed 20k RCT. Εκτελούμε 8 επαναλήψεις του προτεινόμενου Αλγορίθμου και σε κάθε επανάληψη περικόπτεται 14 κεφαλές, επομένως κόβουμε 112 από τις 144 κεφαλές, που είναι περίπου το 80% των κεφαλών αυτο-προσοχής του BERT. Τα αποτελέσματα αυτού του πειράματος μπορούν να φανούν στον Πίνακα 5.

Μια πιο λεπτομερής εικόνα αυτού του πειράματος μπορεί να περιγραφεί από την Εικόνα 4, όπου κόβουμε 14 κεφαλές σε κάθε επανάληψη και συνολικά κόβουμε 112 κεφαλές.

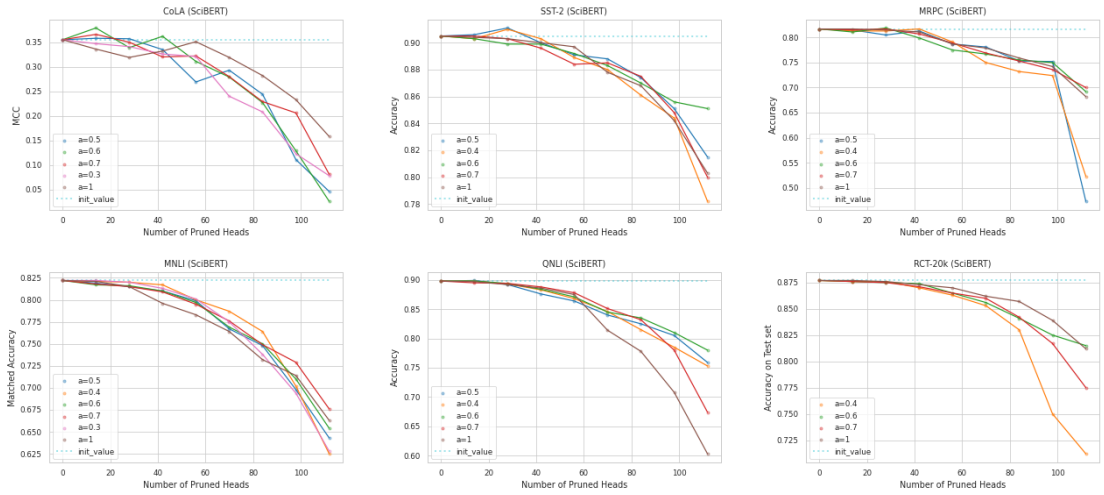


Figure 4. *Αποτελέσματα στα σύνολα δεδομένων PubMed 20k RCT και SciERC. Εκτελούμε πειράματα με διαφορετικές τιμές της υπερπαραμέτρου* α *και καταγράφουμε τα αποτελέσματα του κομμένου μοντέλου στο σύνολο ανάπτυξης. Κόβουμε* 14 *κεφαλές σε κάθε βήμα περικοπής. Για* $\alpha = 1$ *η μεθοδολογία μας είναι ισοδύναμη με την συγκρινόμενη μεθοδολογία του Michel [1].*

Σχετικά με τα προβλήματα του GLUE γνωρίζουμε ότι έχουν μικρή λεξιλογική επικάλυψη με τα

a	MNLI	QNLI	MRPC	SST-2	CoLA
1	0.817	0.820	0.773	0.903	0.329
0.6	**0.824**	0.874	0.778	0.913	0.334
0.5	0.819	0.876	0.607	0.917	**0.435**
0.4	0.809	**0.880**	**0.783**	**0.918**	0.312

Table 6. *Αποτελέσματα στα προβλήματα του GLUE.*

επιστημονικά πεδία που έχει προεκπαιδευτεί το SciBERT και γι' αυτό περιμένουμε ότι η καλύτερη μεθοδολογία περικοπής θα προκύψει με τιμή της υπερπαραμέτρου α μεγαλύτερη από 0.5. Πράγματι, αυτό επιβεβαιώνεται από τα παραπάνω πειράματα. Επιπλέον, σχετικά με το PubMed 20k RCT που είναι ένα επιστημονικό σύνολο, βλέπουμε ότι η καλύτερη τιμή προκύπτει για $\alpha = 0.6$, που δείχνει ότι η καλύτερη στρατηγική περικοπής προκύπτει όταν δίνει μεγαλύτερο βάρος στο μετεκπαιδευμένο μοντέλο.

0.6.3 Κερδίσαμε το Λαχείο;

Εμπνευσμένη από την "Υπόθεσης Τυχερού Δελτίου" (Lottery Ticket Hypothesis) [12] εκτελούμε πειράματα για τα εξείς 5 προβλήματα του GLUE: MNLI, QNLI, MRPC, SST-2 and CoLA. Αρχικά, για κάθε πρόβλημα εκτελούμε τον προτεινόμενο αλγόριθμο για διαφορετικές τιμές του α, το οποίο εκτελεί διαφορετικά σύνολα από μάσκες κεφαλών. Ύστερα, εφαρμόζουμε αυτό το σύνολο μασκών στο "bert-base-uncased" και έτσι έχουμε ένα προεκπαιδευμένο μοντέλο στο οποίο 112 κεφαλές έχουν μαρκαριστεί. Μετά μετεκπαιδεύουμε το μοντέλο στο αντίστοιχο πρόβλημα του GLUE με τις μετρικές που φαίνονται στον Πίνακα 1.

Τα πειράματα μπορούν να περιγραφούν στον Πίνακα 5.7. Η προσέγγισή μας κερδίζει τα μοντέλα που έχουν μαρκαριστεί με σύνολο μασκών που έχει προκύψει από την τεχνική του Michael [1], παρόλο που κόβεται το 80% των κεφαλών αυτο-προσοχής το αντίκτυπο στην απόδοση του μοντέλου έχει μια πτώση μεταξύ $2-4\%$. Έτσι, μπορούμε να παρατηρήσουμε ότι η χρήση του προεκπαιδευμένου και του μετεκπαιδευμένου δικτύου βοηθάει την επιλογή μασκών για την διαδικασία της "Υπόθεσης Τυχερού Δελτίου".

0.6.4 Επαναλαμβανόμενη Δομική Περικοπή

Σε αυτή την ενότητα εκτελούμε το πείραμα της Επαναλαμβανόμενης Δομικής Περικοπής. Αυτή η προσέγγιση βελτιώνει την μελέτη της "Υπόθεσης Τυχερού Δελτίου" επειδή είναι η πρώτη φορά που προτείνεται μια επαναληπτική διαδικασία για την εφαρμογή δομικής περικοπής. Ταυτόχρονα, γενικεύεται η επαναληπτική προσέγγιση του Chen et al. [16].

Σε αυτά τα πειράματα εκτελούμε μια επαναληπτική τεχνική περικοπής όπου αρχικά μετεκπαιδεύουμε το BERT μοντέλο στα τελικά προβλήματα. Μετά εντοπίζουμε τις 14 κεφαλές με το μικρότερη επίδοση στην ευριστική που προτείνουμε για $\alpha = 1$. Μετά παίρνουμε το προεκπαιδευμένο μοντέλο και κόβουμε αυτές τις 14 κεφαλές και μετεκπαιδεύουμε το μοντέλο. Έτσι, το τελικό μοντέλο έχει 14 λιγότερες κεφαλές και έχει εκπαιδευτεί μόνο μια φορά. Μπορούμε να συνεχίσουμε την διαδικασία εντοπίζοντας 14 κεφαλές με τη μικρότερη επίδοση στην ευριστική που προτείνουμε. Έτσι, παίρνουμε το προεκπαιδευμένο μοντέλο και κόβουμε αυτές τις $14 + 14$ κεφαλές και μετά το μετεκπαιδεύουμε.

Συγκρίνουμε τα αποτελέσματα μας με την μη επαναλαμβανόμενη τεχνική η οποία περικόπτει το $x\%$ των κεφαλών του προεκπαιδευμένου μοντέλο και μετά μετεκπαιδεύτηκε. Έτσι, το τελικό μοντέλο αποτελείται από $100 - x\%$ κεφαλές αυτο-προσοχής.

Με την Επαναλαμβανόμενη Δομική Περικοπή η συνολική απόδοση του μοντέλου είναι μεγαλύτερη από την μη επαναλαμβανόμενη. Αυτό μπορεί να φανεί από την εικόνα Figure 5 όπου εκτελέσαμε αυτό

το πείραμα σε 2 προβλήματα GLUE: MNLI και QNLI. Αυτή η προσέγγιση δείχνει ότι υπάρχει ένα δίλημμα (trade-off) μεταξύ επίδοσης και υπολογιστικού κόστους.

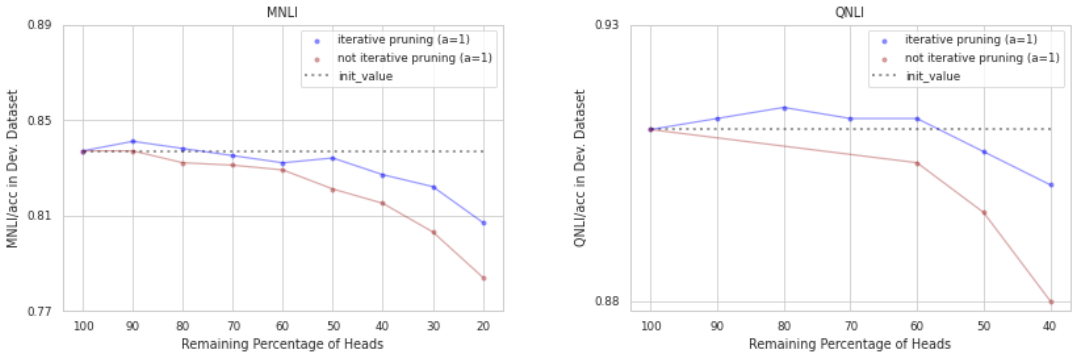


Figure 5. *Επαναλαμβανόμενη Δομική Περικοπή στο MNLI και στο QNLI με* $\alpha = 1$

0.6.5 Περικοπή σε Μοντέλα Διαφορετικής Τροπικότητας

Σε αυτή την ενότητα κόβουμε κεφαλές αυτο-προσοχής σε ένα μοντέλο διαφορετικής τροπικότητας (modality). Πιο συγκεκριμένα, εξετάζουμε την τροπικότητα της φωνής μέσω του μοντέλου wav2vec 2.0 [26]. Είναι η πρώτη φορά που γίνεται περικοπή κεφαλών αυτο-προσοχής στο wav2vec 2.0.

Το προεκπαιδευμένο μοντέλο wav2vec 2.0. έχει μετεκπαιδευτεί στο σύνολο δεδομένων Timit [2] και η επίδοση του καταγράφεται στην μετρική του Word Error Rate. Τα αποτελέσματα μπορούν να φανούν από την Εικόνα 6. Έτσι, μπορούμε να δούμε ότι η τεχνική μας μπορεί να εφαρμοστεί και σε διαφορετικές τροπικότητες, και μάλιστα σε βαθιά επίπεδα περικοπής επιτυγχάνει καλύτερα αποτελέσματα από την μεθοδολογία σύγκρισης.

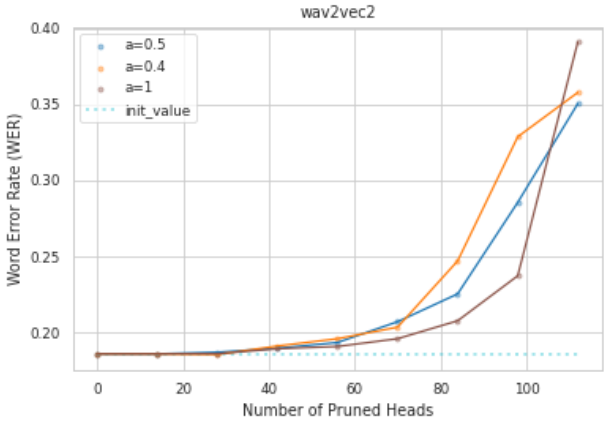


Figure 6. *Το μοντέλο wav2vec 2.0. έχει μετεκπαιδευτεί στο σύνολο δεδομένων Timit [2] και η επίδοση του καταγράφεται στην μετρική του Word Error Rate.*

Chapter 1

Introduction

1.1 From Artificial Intelligence To Deep Learning

The terms Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) are usually confused. Based on the Oxford Dictionary, Artificial Intelligence can be defined as "The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages". Thus, we can conclude that the term Artificial Intelligence is a more general term including AI, ML, and DL, and the distinction is based on the technique in which the computer systems can mimic human intelligence.

Artificial Intelligence approaches use hard-code knowledge about the ongoing task. Based on logical inference rules, a computer system can derive new knowledge and reason about statements. This method is known as the knowledge-based approach to artificial intelligence. One drawback of this approach is the difficulty of formally describing all the knowledge based on the given task.

Machine Learning tries to overcome this problem by letting the system extract patterns from raw data. The most naive machine learning method is the Naive Bayes classifier, based on the Bayes' theorem. The performance of these simple machine learning algorithms depends heavily on the representation and the amount of the given data. Many machine learning methodologies, such as the naive Bayes classifier, try to derive a mapping representation from the representation space to the output space. However, mapping is not always the case for machine learning. Another aim of machine learning is to learn the representation itself, and this field is called representation learning. It can be very difficult to extract such high-level, abstract features from raw data. Many of these factors of variation, such as a noise environment, can make the process quite challenging.

Deep Learning solves this central problem in representation learning by introducing representations that are expressed in terms of other simpler representations. Deep learning enables the computer to build complex concepts out of simpler concepts but demands enormous data.

1.2 Motivation

One of the most successful deep learning architectures is called Transformers. Transformers yield performance in many Natural Language Processing tasks, while the training pipeline is computationally demanding and the final model lists millions of parameters. Indeed, Transformer-based models keep growing by orders of magnitude: The 110M parameters of base BERT are now dwarfed by 17B parameters of Turing-NLG [22], which is dwarfed by 175B of GPT-3 [23]. This trend raises concerns about computational complexity of self-attention, environmental issues [34] [35], fair comparison of architectures [36], and reproducibility.

Human language is incredibly complex and would perhaps take many more parameters to describe fully [37], but the current models do not make good use of the parameters they already

have. Voita et al. [14] showed that all but a few Transformer heads could be pruned without significant losses in performance. For BERT, Clark et al. [27] observe that most heads in the same layer show similar self-attention patterns (perhaps related to the fact that the output of all self-attention heads in a layer is passed through the same MLP), which explains why Michel et al. [1] were able to reduce most layers to a single head.

Depending on the task, some BERT heads/layers are not only redundant [38], but also harmful to the downstream task performance. Positive effect from head disabling was reported for machine translation [1], abstractive summarization [39], and GLUE tasks [13]. Additionally, Tenney et al. [40] examine the cumulative gains of their structured probing classifier, observing that in 5 out of 8 probing tasks, some layers cause a drop in scores (typically in the final layers). From the aspect of unstructured pruning, Gordon et al. [28] find that 30–40 percent of the weights can be pruned without impact on downstream tasks.

In general, larger BERT models perform better [41] , [42] but not always: BERT-base outperformed BERT-large on subject-verb agreement [43] and sentence subject detection [44]. Given the complexity of language and amounts of pre-training data, it is not clear why BERT ends up with redundant heads and layers. Clark et al. [27] suggest that one possible reason is the use of attention dropouts, which causes some attention weights to be zeroed-out during training.

Given the above evidence of **overparameterization**, it does not come as a surprise that BERT can be efficiently compressed with minimal accuracy loss, which would be highly desirable for real-world applications.

Compression can be achieved with different techniques, such as Knowledge Distillation [45], Pruning [46] and Neural Network Quantization [47]. In this study, we explore Pruning as a compression technique and, more specifically, Structured Pruning.

The main reason we study Structured Pruning rather than Magnitude pruning is that adaptive sparse matrix multiplication has shown promising results on GPUs but is not yet applied in silicon, and thus structured pruning is the only way of actually "accelerating" BERT. On the other hand, accelerating unstructured sparse matrix multiplication is an active area of research in which recent progress has been made. Bank-balanced sparsity (which is closely related to unstructured sparsity) achieves near-ideal speed-ups while requiring a minimal deviation from unstructured sparsity [48].

Moreover, through structured pruning, we investigate the effect of structural components in the neural network. We understand their functionality and their usefulness in the whole network, and through this study, we can achieve better fine-tuning techniques.

1.3 Research Contributions

In this study, we examine a structured pruning approach for BERT-based architectures that implies that in transfer learning, the final model is the result of a fine-tuning process of a pre-trained model. Thus, we consider both the pre-trained and the fine-tuned model to prune attention heads. Furthermore, we study this method through the Lottery Ticket Hypothesis, where we see that considering both the pre-trained and the fine-tuned model outperforms the approach which only considers the fine-tuned model. Moreover, we propose a better application of the Lottery Ticket Hypothesis in structured pruning named "Iterative Structured Pruning". Finally, we examine our technique on another modality, more precisely in Automatic Speech Recognition through wav2vec 2.0, and we see corresponding results.

1.4 Thesis Outline

Chapter 2: From Machine Learning to Deep Learning, provides background knowledge to set the stage for the subsequent chapters. First, we provide an overview of technical information that is relevant in order to understand the contents of this thesis. Next, we introduce the reader to machine learning together with its most elementary methods. We subsequently delve into the machine learning models primarily used in this thesis.

Chapter 3: Natural Language Processing, presents the natural language processing background needed to understand this thesis. After briefly presenting popular natural language processing tasks, language modeling is presented, initially in the form of an-gram model based on the Markov assumption and then as a recurrent neural network. Then, transfer learning methods that are currently used to train natural language processing models are explained.

Chapter 4: Compression of Deep Learning Models, is a short survey that introduces compression techniques in deep learning models. Firstly, the problem description is introduced, and the Lottery Ticket Hypothesis is explained. Then we focus on different pruning approaches for Transformer-based models, while we also present similar techniques on Computer Vision. Finally, critical aspects of Explainable AI are presented as a promising pruning methodology.

Chapter 5: Back to the Future: A transfer learning approach for structured pruning, presents a novel structured pruning technique for Deep Learning Language Models, such as BERT. This approach considers both the pre-trained and the fine-tuned model and outperforms other structured pruning methodologies. Also, we show that this method produces a better set of head masks for the Lottery Ticket Hypothesis; if this set is used in the pre-trained model and this model is fine-tuned, it will produce a model with significant performance and sparsity. Finally, in this chapter, we examine an iterative structured pruning algorithm for performing the Lottery Ticket Hypothesis, and we apply our methodology to a different modality.

Chapter 6: Conclusions, contains our conclusion, summarizing our findings and providing an outlook into the future work.

Chapter 2

From Machine Learning to Deep Learning

2.1 Introduction

The terms Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) are usually confused. Based on the Oxford Dictionary, Artificial Intelligence can be defined as "The theory and development of computer systems able to perform tasks normally requiring human intelligence, such as visual perception, speech recognition, decision-making, and translation between languages". Thus, we can conclude that the term Artificial Intelligence is a more general term including AI, ML, and DL, and the distinction is based on the technique in which the computer systems can mimic human intelligence.

Artificial Intelligence approaches use hard-code knowledge about the ongoing task. Based on logical inference rules, a computer system can derive new knowledge and reason about statements. This method is known as the knowledge-based approach to artificial intelligence. One drawback of this approach is the difficulty of formally describing all the knowledge based on the given task.

Machine Learning tries to overcome this problem by letting the system extract patterns from raw data. The most naive machine learning method is the Naive Bayes classifier, based on the Bayes' theorem. The performance of these simple machine learning algorithms depends heavily on the representation and the amount of the given data. Many machine learning methodologies, such as the naive Bayes classifier, try to derive a mapping representation from the representation space to the output space. However, mapping is not always the case for machine learning. Another aim of machine learning is to learn the representation itself, and this field is called representation learning. It can be very difficult to extract such high-level, abstract features from raw data. Many of these factors of variation, such as a noise environment, can make the process quite challenging.

Deep Learning solves this central problem in representation learning by introducing representations that are expressed in terms of other simpler representations. Deep learning enables the computer to build complex concepts out of simpler concepts but demands enormous data.

In this section, we are going to discuss Machine Learning and Deep Learning Basics.

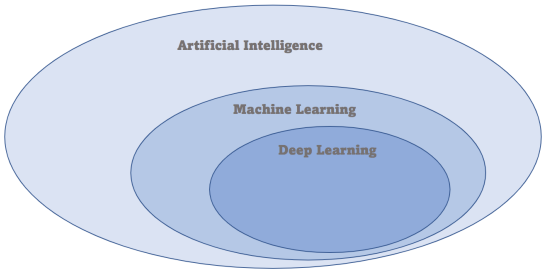


Figure 2.1. *Artificial Intelligence, Machine Learning and Deep Learning Domains*

2.2 Learning Classification

Before classifying the types of learning in machine learning, a proper definition of "learning" should be provided. Indeed, Mitchell [] provides generic, well-suited, definition for learning: "A computer program is said to learn from experience (E) concerning some class of tasks (T) and performance measure (P), if its performance at tasks in T, as measured by P, improves with experience E." So, more specifically in machine learning we train our model with data in order to gain experience for a given task. Hopefully, after the training process, we will notice performance improvement. In this section, we are going to classify learning, and we can define 14 different learning types:

1. Learning Problems
   (a) Supervised Learning
   (b) Unsupervised Learning
   (c) Reinforcement Learning
2. Hybrid Learning Problems
   (a) Semi-Supervised Learning
   (b) Self-Supervised Learning
   (c) Multi-Instance Learning
3. Statistical Inference
   (a) Inductive Learning
   (b) Deductive Inference
   (c) Transductive Learning
4. Learning Techniques
   (a) Multi-Task Learning
   (b) Active Learning
   (c) Online Learning
   (d) Transfer Learning
   (e) Ensemble Learning

2.2.1 Supervised Learning

In supervised learning, there are input variables X and an output variable Y. The goal is to build an algorithm that learns the mapping function from the input to the output.

$$Y = f(X)$$

During training, both X and their corresponding y are provided; thus, the training data are labeled. At inference, we expect that the mapping function can successfully predict the output for every given x provided from the same distribution as the training sample.

Moreover, supervised learning can be classified into two categories, based on the nature of the output y:

1. Classification

2. Regression

In classification, the output y is the class label of the input data x, while in regression, the output is a predicted numerical value.

2.2.2 Unsupervised Learning

In other machine learning problems, the training data consists of input vectors x without any corresponding target values. So, unsupervised learning aims to find patterns where the target values are either not observable or infeasible to obtain. A large subclass of unsupervised tasks is the problem of clustering. Clustering refers to grouping observations together so that members of a common group are similar and different from members of other groups. Another exciting class of unsupervised tasks is generative modeling. Generative models are models that imitate the process that generates the training data. A good generative model would generate new data that resemble the training data in some sense. This type of learning is unsupervised because the process that generates the data is not directly observable – only the data itself is observable.

2.2.3 Self-Supervised Learning

Self-Supervised learning is proposed for utilizing unlabeled data with the success of supervised learning. Producing a dataset with suitable labels is expensive, while unlabeled data is being generated all the time. The motivation of Self-Supervised Learning is to make use of a large amount of unlabeled data.

The main idea of Self-Supervised learning is to generate the labels from unlabeled data, according to the structure or characteristics of the data itself, and then train on this unsupervised data in a supervised manner. Self-Supervised learning is wildly used in representation learning to make a model learn the latent features of the data. This technique is often employed in both natural language processing, and computer vision [49].

2.2.4 Transfer Learning

The ideal scenario of machine learning is that there are abundant labeled training instances, which have the same distribution as the test data. However, collecting sufficient training data is often expensive, time-consuming, or even unrealistic in many scenarios. Semi-supervised learning can partly solve this problem by relaxing the need for mass labeled data. Typically, a semi-supervised approach only requires a limited number of labeled data, and it utilizes a large amount of unlabeled data to improve the learning accuracy. However, unlabeled instances are also challenging to collect in many cases, making the resultant traditional models unsatisfactory.

Transfer learning, which focuses on transferring knowledge across domains, is a promising machine learning methodology for solving the above problem. The concept of transfer learning may initially come from educational psychology. According to the generalization theory of transfer, as proposed by psychologist C.H. Judd, learning to transfer is the result of the generalization of experience. It is possible to realize the transfer from one situation to another if a person generalizes his experience. According to this theory, the transfer prerequisite is the interconnection between the learning activities. In practice, a person who has learned the violin can learn the piano faster than others since both the violin and the piano are musical instruments and may share some common knowledge.

Inspired by human beings' capabilities to transfer knowledge across domains, transfer learning aims to leverage knowledge from a source domain to improve the learning performance or minimize the number of labeled examples required in a target domain. However, it is worth mentioning that

the transferred knowledge does not always positively impact new tasks. If there is little in common between domains, knowledge transfer could be unsuccessful.

We will provide a more formal definition of the terms mentioned above; Domain, Target, and Transfer Learning.

Domain: A domain $\mathcal{D}$ is composed of two parts, i.e., a feature space $\mathcal{X}$ and a marginal distribution $P(X)$. In other words, $\mathcal{D} = \{\mathcal{X}, P(X)\}$. And the symbol X denotes an instance set, which is defined as $X = \{\mathbf{x} \mid \mathbf{x}_i \in \mathcal{X}, i = 1, \ldots, n\}$

Task: A task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a decision function f, i.e., $\mathcal{T} = \{\mathcal{Y}, f\}$. The decision function f is an implicit one, which is expected to be learned from the sample data.

Transfer Learning: Given some observation(s) corresponding to $m^S \in \mathbb{N}^+$ source domain(s) and task(s) (i.e., $\left\{(\mathcal{D}_{S_i}, \mathcal{T}_{S_i}) \mid i = 1, \ldots, m^S\right\}$), and some/an observation(s) about $m^T \in \mathbb{N}^+$ target domain (s) and task (s) (i.e., $\left\{\left(\mathcal{D}_{T_j}, \mathcal{T}_{T_j}\right) \mid j = 1, \ldots, m^T\right\}$), transfer learning utilizes the knowledge implied in the source domain(s) to improve the performance of the learned decision functions $f^{T_j}(j = 1, \cdots, m^T)$ on the target domain(s).

If m^S equals 1 , the scenario is called single-source transfer learning. Otherwise, it is called multi-source transfer learning. Besides, m^T represents the number of the transfer learning tasks where in the majority of scenarios $m^T = 1$ [50].

2.3 Machine Learning Concepts

2.3.1 Loss Function

The goal of any Supervised Learning algorithm is to return a function $f()$ which accurately matches the input examples to the corresponding labels. To quantify the loss (error) of the model, a Cost Function is used that predicts $\hat{y}$ when the actual label is y. Usually, the Cost Function $L(\hat{y}, y)$ assigns a numeric value to the predicted output $\hat{y}$ given the actual output y. It must have an infimum, which means that the lower the error value, the better the prediction. Function parameters are set in order to minimize L loss in the training examples.

Given a train set $(x_{1:n}, y_{1:n})$, a cost function L per sample and a function $f(x; \theta)$, we define the total loss as the average loss on all training data:

$$\mathcal{L}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(f(x; \theta), y_i\right)$$

The goal is to find the optimal parameters θ that minimize the total error:

$$\hat{\theta} = \arg_\theta \min \mathcal{L}(\theta) = \arg_\theta \min \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(f(x; \theta), y_i\right)$$

Some standard cost functions are the following:

Mean Squared Error (MSE): MSE calculates the mean squared prediction error:

$$J(\partial) = \frac{1}{n} \sum_{i=1}^{n} \left(Y_i - P_i\right)^2$$

Where the prediction error is the difference between the true value (Y_i) and the predicted value (P_i) for an instance, and ∂ is the parameter vector of the network. MSE is used with regression models.

Mean Absolute Error (MAE): MAE calculates the mean of the absolute prediction error:

$$J(\partial) = \frac{1}{n}\sum_{i=1}^{n}|Y_i - P_i|$$

Where Y_i is the true value and P_i is the predicted value for an instance, and ∂ is the parameter vector of the network.

Cross Entropy: Cross-entropy loss function uses the concept of cross-entropy. Cross-entropy is mathematically defined as:

$$H(p, q) = -\sum_{k} p_k \log q_k$$

Where p and q are the true and the predicted probability distributions, respectively, the more the two distributions differ, the higher the value of the cross-entropy. The cross-entropy loss function is widely used in classification problems. Based on the definition of cross-entropy, the goal of the Cross-entropy loss function is to minimize the cross-entropy between the model's distribution and the distribution of the given data.

2.3.2 Underfitting and Overfitting

The central challenge in machine learning is that the proposed algorithm should perform well on new, previously unseen inputs, not just those our model is trained. The ability to perform well on previously unobserved inputs is called generalization. Typically, when training a machine learning model, we have access to a training set; we can compute some error measure on the training set, called training error; and we reduce this training error. So far, what we have described is simply an optimization problem.

What separates machine learning from optimization is that we want the generalization error, also called the test error, to be below. The generalization error is defined as the expected value of the error on new input. Here the expectation is taken across different possible inputs, drawn from the distribution of inputs we expect the system to encounter in practice. We typically estimate the generalization error of a machine learning model by measuring its performance on a test set of examples that were collected separately from the training set.

The factors determining how well a machine learning algorithm will perform its ability to: make the training error small and make the gap between training and test error small. These two factors correspond to the two central challenges in machine learning: underfitting and overfitting. Underfitting occurs when the model cannot obtain a sufficiently low error value on the training set. Overfitting occurs when the gap between the training error and test error is too large. So, in machine learning, we are trying to find a good trade-off of training error and the gap between training and test error as it can be described by the Figure 2.2.

2.3.3 Regularization, Dropout and Pruning

We want to overcome the problem of overfitting and improve the generalization. Our modern ideas about improving the generalization of machine learning models are refinements of thought dating back to philosophers at least as early as Ptolemy. Many early scholars invoke a principle of parsimony that is now most widely known as Occam's razor (c. 1287–1347). This principle states that among competing hypotheses that explain known observations equally well, we should choose the "simplest" one. This idea was formalized and made more precise in the twentieth century by the founders of statistical learning theory.

Regularization is the most common way to mitigate overfitting and apply Occam's razor on machine learning problems. To face the potential loss of generalization, we impose restrictions

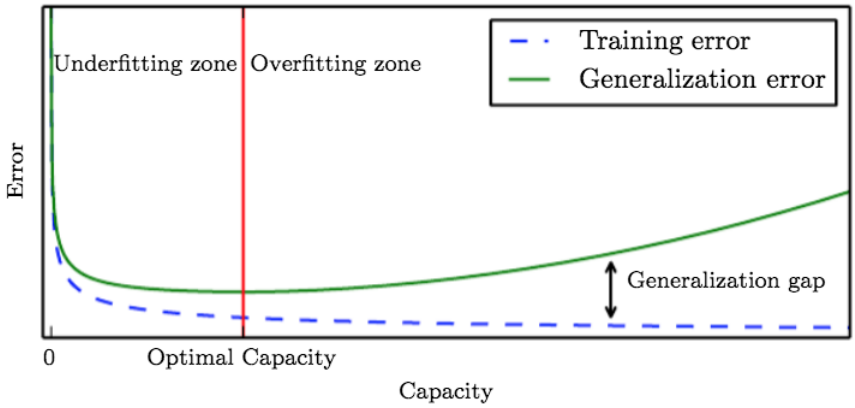


Figure 2.2. *Training and test errors behave differently. At the left end of the graph, training error and generalization error are both high. This is the underfitting regime. As we increase capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the overfitting regime, where capacity is too large, above the optimal capacity. Source: [3]*

on the form of the solution by forcing the model to choose the smallest - in order of parameters- solution. This is done by implying a term in the loss equation then penalizing the size of the model. Thus, the loss function takes the following form:

$$\hat{\theta} = \arg\min_{\theta} \mathcal{L}(\theta) = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L\left(f\left(\mathbf{x}_i; \theta\right), y_i\right) + \lambda R(\theta)$$

The regularization term considers the parameter values and scores their complexity. We then look for values that have both a low loss and low complexity. What regularization inherently intends to do is penalize complex models and favor simpler ones. λ is a value that must be set manually, based on the classification performance on a development set (called hyperparameter). The Regularizers R measure the norms of the parameter matrices and opt for solutions with low norms. The two most common regularization norms are L_2 and L_1, and the most common technique is dropout.

L_2 regularization. R takes the form of the standard Euclidean norm (L_2-norm) of the parameters, trying to keep the sum of the squares of the parameter values low. Large model weights $\mathbf{W}_{[i,j]}$ will be penalized, since they are considered "unlikely". L_2 is often referred to as weight decay. As one can observe, high weights are severely penalized, but weights with small values are only negligibly affected.

$$R_{L_2}(\mathbf{W}) = \|\mathbf{W}\|_2^2 = \sum_{i,j} \left(\mathbf{W}_{[i,j]}\right)^2$$

L_1 regularization. The L_1 regularizer punishes uniformly low and high values and intends to decrease all non-zero parameter values towards zero. So, it encourages sparse solutions or else models with many parameters with a zero value. The L_1 regularizer is also called a sparse prior or lasso.

$$R_{L_1}(\mathbf{W}) = \|\mathbf{W}\|_1 = \sum_{i,j} \left|\mathbf{W}_{[i,j]}\right|$$

Dropout. An effective technique for preventing neural networks from overfitting the training samples is dropout training. Dropout is designed to prevent the network from learning to rely on specific weights. It randomly sets to zero (drops) half of the neurons in the network (or in a specific

layer) in each training example, in the stochastic-gradient training. The dropout technique is one of the key factors contributing to the robust results of neural networks.

Pruning Another way to prevent overfitting is pruning. Pruning involves removing connections between neurons or entire neurons, channels, or filters from a trained network, which is done by zeroing out values in its weights matrix or removing groups of weights entirely. We will discuss pruning with more details in Chapter 4.

2.3.4 Gradient descent

In this section, we will discuss a non-optimal but efficient way of minimizing the loss function. A gradient (at a point) is the slope of the tangent to the function at that point. It points to the direction of the most significant increase of the function. Gradient-based methods minimize the objective function $\mathcal{L}(\theta)$ by repeatedly computing an estimate of the loss $\mathcal{L}$ over the training set, computing the gradients of the parameters θ of the model concerning the loss estimate and updating the parameters in the opposite direction of the gradient.

Gradient descent (GD) is one of the most popular algorithms to perform optimization in neural networks. It computes the gradient of the cost function concerning the parameters θ for the entire dataset. The learning rate (η) is a hyperparameter that controls the extent to which the model parameters are adjusted concerning the loss gradient. GD is formally defined as:

$$\theta = \theta - \eta \nabla_\theta J(\theta)$$

Stochastic Gradient Descent (SGD) in contrast performs a parameter update for each training example $\mathbf{x}_i$ and label y_i :

$$\theta = \theta - \eta \nabla_\theta J\left(\theta; \mathbf{x}_i; y_i\right)$$

Gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is, therefore, usually much faster and can also be used to learn online.

Vanilla gradient descent, however, does not guarantee good convergence. The learning rate needs to be carefully tuned, as a small value leads to slow convergence, while a very large value can hinder convergence and cause the loss function to diverge or to fluctuate around the minimum as shown in Figure 2.3. Moreover, the same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, it is possible that we do not want to update them to the same extent, but we instead want to perform larger updates for rarely occurring features. Finally, a key challenge of minimizing highly non-convex error functions, common for neural networks, is avoiding getting trapped in local minima, as shown in Figure 2.4.

2.4 Machine Learning Models

2.4.1 Linear Regression

Linear Regression is a Supervised Learning Problem and solves a regression problem. In other words, the goal is to build a system that can take a vector $\boldsymbol{x} \in \mathbb{R}^n$ as input and predict the value of a scalar $y \in \mathbb{R}$ as its output. The output of linear regression is a linear function of the input. Let $\hat{y}$ be the value that our model predicts y should take on. We define the output to be:

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$$

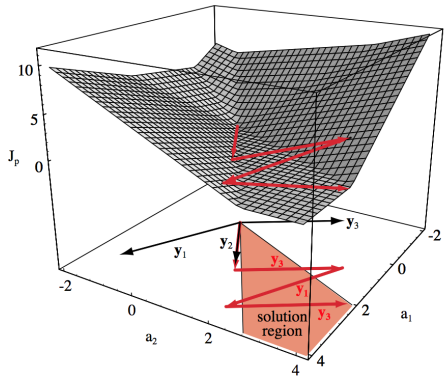


Figure 2.3. *Large learning rate leads the loss function to fluctuates around the minimum. Source: [4]*

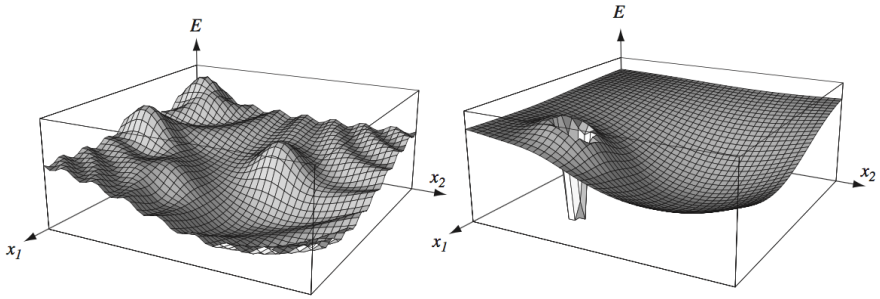


Figure 2.4. *Loss function's landscapes could be full of local minima, which potentially could lead to a suboptimal solution by GD Source: [4]*

where $\boldsymbol{w} \in \mathbb{R}^n$ is a vector of parameters.

Parameters are values that control the behavior of the system. In this case,w_i is the coefficient that we multiply by feature x_i before summing up the contributions from all the features. We can think of w as a set of weights that determine how each feature affects the prediction. An in order to achieve an optimal fit of the model to the training data, we try to minimize $\text{MSE}_{\text{train}}$:

$$\text{MSE}_{\text{train}} = \frac{1}{m} \sum_i (\hat{\boldsymbol{y}} - \boldsymbol{y})_i^2$$

To minimize MSE $_{\text{train}}$, we can simply solve for where its gradient is $\mathbf{0}$:

$$\nabla_{\boldsymbol{w}} \text{MSE}_{\text{train}} = 0$$

$$\Rightarrow \boldsymbol{w} = (\boldsymbol{X}\boldsymbol{X})^{-1} \boldsymbol{X}\boldsymbol{y}$$

So, for Linear Regression, we have a close form to describe the optimal solution. However, it is preferred to calculate the w with gradient descent because this method is computationally intensive both in time and memory complexity. Moreover, GD is capable of parallelization. Finally, we prefer GD, because this method does not provide a solution when the data is non-linear separable, because then the matrix $(\boldsymbol{X}\boldsymbol{X})^{-1}$ is not invertible.

2.4.2 Logistic Regression

When it comes to classification, we are determining the probability of an observation to be part of a certain class or not. Therefore, we wish to express the probability with a value between 0 and

1. A simple classification algorithm that generates values of that form is Logistic Regression (LR) classifier.

Suppose we have a simple problem for a binary classifier, as described earlier in the same Section. Let $\mathbf{x}_{i=1:N} = \mathbf{x}_1, \ldots, \mathbf{x}_N$ be the input vectors where $_i \in \{0, 1\}$. The activation of the LR classifier is determined by applying a sigmoid function over the fitted line to get the final classification decision:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The activation of LR for a given vector $\mathbf{x}$ is defined as follows:

$$h_{\mathbf{w}}(\mathbf{x}) = \sigma\left(\mathbf{w}^T\mathbf{x}\right) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

The loss function to be minimized while fitting the LR to the training data is the following:

$$\frac{1}{2}\|\mathbf{w}\| + C\sum_{i=1}^{N}\log\left(\exp\left(-y_i\left(\mathbf{w}^T\mathbf{x}_i + \mathbf{b}\right)\right) + 1\right)$$

where $C > 0$ and b represent the coefficients of penalization of wrong classifications and LR is a highly efficient technique that does not require extensive computational resources usage. It thus provides a solid baseline for most NLP tasks. An obvious disadvantage is that LR cannot solve non-linear problems since its decision surface is linear. A common practice used to apply LR to multi-class classification is to iteratively treat each pair of classes as a binary classification problem, to which LR is performed.

2.4.3 Other Models

Some worth mentioning machine learning models are the following:

Decision Trees are a type of Supervised Machine Learning where the data is continuously split according to a specific parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the outcomes. Furthermore, the decision nodes are where the data is split.

Conditional random fields (CRFs) are a class of statistical modeling methods often applied in pattern recognition and machine learning and used for structured prediction. Whereas a classifier predicts a label for a single sample without considering "neighboring" samples, a CRF can take context into account. The prediction is modeled as a graphical model, which implements dependencies between the predictions. What kind of graph is used depends on the application. For example, linear-chain CRFs are widespread in natural language processing, which implements sequential dependencies in the predictions. In image processing, the graph typically connects locations to nearby and similar locations to enforce that they receive similar predictions.

(Gaussian) Mixture Model is a probabilistic model for representing the presence of subpopulations within an overall population, without requiring that an observed data set should identify the sub-population to which an individual observation belongs. Formally a mixture model corresponds to the mixture distribution representing the probability distribution of observations in the overall population. However, while problems associated with "mixture distributions" relate to deriving the properties of the overall population from those of the sub-populations, "mixture models" are used to make statistical inferences about the properties of the sub-populations given only observations on the pooled population, without subpopulation identity information.

2.5 Deep Learning Models

Deep Learning solves this central problem in representation learning by introducing representations expressed in other simpler representations. Deep learning enables the computer to build complex concepts out of simpler concepts but demands big amount of data. In this section, we are going to discuss some of the most popular Deep Learning Models.

2.5.1 Feedforward Neural Networks

An Artificial Neural Network (ANN) is a biologically inspired computational model patterned after the network of neurons present in the human brain. The area of ANNs has been initially inspired by modeling biological neural systems but has since diverged and become a matter of engineering and achieving good results in Machine Learning tasks. Thus, we first introduce a very brief and high-level description of the biological system that has influenced a large portion of deep learning. The basic computational unit of the brain is a neuron. Billions of neurons can be found in the human nervous system. Figure 2.5 shows the comparison between a biological neuron and its mathematical notation. Each neuron receives input signals from its dendrites and produces output signals along its (single) axon. The axon connects via synapses to the dendrites of other neurons. In the computational model of a neuron, the signals that travel along the axons (e.g., x_0) interact multiplicatively (e.g., $x_0 w_0$) with the dendrites of the other neuron based on the synaptic strength at that synapse (e.g., w_0). The idea is that the synaptic strengths (the weights w) are learnable and control the strength of influence of one neuron on another. In the basic model, the dendrites carry the signal to the cell body, where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In the computational model, we assume that only the frequency of the firing communicates information. We thus model the neuron's firing rate with an activation function f, which represents the frequency of the spikes along the axon. A standard activation function is the sigmoid function σ, since it takes a real-valued input and squashes it to a range between 0 and 1.

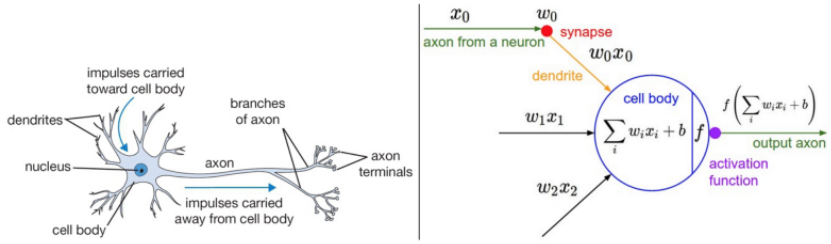


Figure 2.5. *Perceptron Structure. Source: cs231n.github.io*

In order to learn complex non-linear functions, architectures that combine several artificial neurons can be designed and implemented. Such architectures are called Multi-Layer Perceptrons (MLPs). A multilayer perceptron (MLP) is a class of feedforward artificial neural networks (FFNN). An MLP consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer. Except for the input nodes, each node is a neuron that uses a non-linear activation function. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable. In Figure 2.6 we visualize the difference between a simple neural network and a deep neural network (such as an MLP or an FFNN). A deep neural network consists of more than one hidden layer.

Each neural network is composed of the following layers:

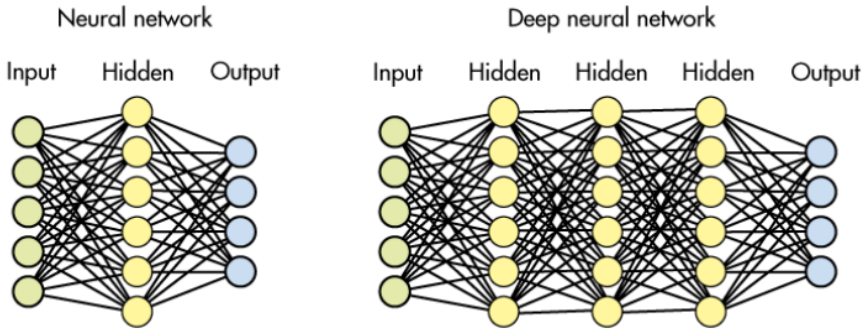


Figure 2.6. *Multilayer perceptron (MLP) Structure. Source: electronicdesign*

- **Input layer**. This layer accepts the input data. It provides information from the outside world to the network without any further computation. Nodes pass on the information to the hidden layer.
- **Hidden layer(s)**. More than one hidden layer can be used to preprocess the inputs obtained by the previous layer. They extract the required features from the input data. When moving to higher hidden layers, higher-level features are constructed.
- **Output layer**. After data preprocessing, a decision is made by the network in this layer.

2.5.2 Recurrent Neural Networks

A recurrent neural network (RNN) is a type of artificial neural network where connections between units form a directed cycle, and it was proposed in the 1980s. This creates an internal state of the network, which allows it to exhibit dynamic behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process and work on arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented connected handwriting recognition, speech recognition, Natural language processing and Machine Translation.

RNNs are especially useful with sequential data because each neuron can use its internal memory to maintain information about the previous input. Another way to think about RNNs is that they have a memory state which captures information about what has been calculated so far. Figure 2.7 illustrates the architecture of a rolled-up recurrent neural network. A recurrent neural network can be considered multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop as it can be described from Figure 2.8.

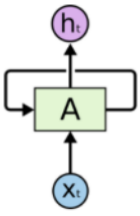


Figure 2.7. *A rolled up RNN where X_t is the input vector containing sequences of characters of a word while h_t is as output vector. Source: colah.github.io*

Considering an example from a natural language such as, "I had washed my house" is much more different than "I had my house washed". This allows the network to gain a deeper understanding of the language statements. This is important to note because reading through a sentence, even

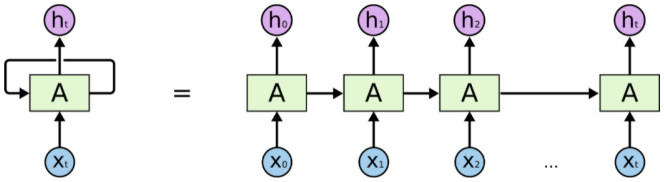


Figure 2.8. *An unrolled up RNN where X_t is the input vector containing sequences of characters of a words while h_t is as output vector. Source: colah.github.io*

as a person, you are picking up each word's context from the words before or after it. An RNN has loops in them that allow information to be carried across neurons while reading input.

In these diagrams, X_t is some input, A is a part of the RNN, and h_t is its output. Essentially you can feed in language words from the sentence or even characters from a string as X_t, and through the RNN, it will result with a h_t. The goal is to use h_t as output and compare it to test data (which is usually nothing but a small subset of the original data). Then we will get error rate information. After comparing the output to the test data, with error rate in hand, we can use Back Propagation Through Time (BPTT) which back checks through the network and adjusts the weights based on error rate and makes it learn to get a good result.

Vanilla RNNs. The RNN first takes the x_0 from the sequence of input, and it outputs h_0 (hidden state). The hidden state h_0 along with the next input x_1 is the input for the next step. Accordingly, h_1 along with x_2 is the input for the next step and so on. So, the RNN remembers the context of the input it has already seen while training. Formally, at each time step t, the equations that describe the function of the RNN are:

$$\mathbf{h_t} = f_h\left(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x_t} + \mathbf{b_h}\right)$$
$$\mathbf{y_t} = f_y\left(\mathbf{W}_{yh}\mathbf{h_t} + \mathbf{b_y}\right)$$

where h_t is the hidden state at time step t, x_t is the input vector at time step t, y_t is the output vector at time step t, b_h is the bias for h, by is the bias for y and f_x, f_h are the activation functions for x and h respectively. They are three separate matrices of weights: W_{hx} (input-to-hidden weights), W_{hh} (hidden-to-hidden), and W_{yh} (hidden-to-output).

Long Short-Term Memory (LSTM). RNNs can handle context from the beginning of the statement, which will allow more accurate predictions of a word at the end of a statement. In practice, this is not necessarily true for all types of RNNs, since RNNs are actually limited to looking back only a few steps. This is a significant reason why RNNs need to be used with a Long Short Term Memory (LSTM) regime, which introduced by Hochreiter and Schmidhuber [51]. Adding the LSTM to the network is like adding a memory unit inside the network that can remember context from the very beginning of the input. So, if the sentence is of 10 words and we want to predict 11th word, all 10 words are being processed by RNNs and their weights on each step are being saved using LSTM and the probability of 11th word being predicted accordingly. Another problem of Vanilia RNN's is the so-called problem of vanishing and exploding gradient through BPTT.

A memory cell is used in addition to the hidden layer to pass information that might not be used in prediction. These memory units allow for RNNs to give much more accurate results. These memory units allow the network to remember the context across inputs. LSTM is denoted in Figure 2.9

Given a sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_n$ of vectors of an input sequence of length n, for vector

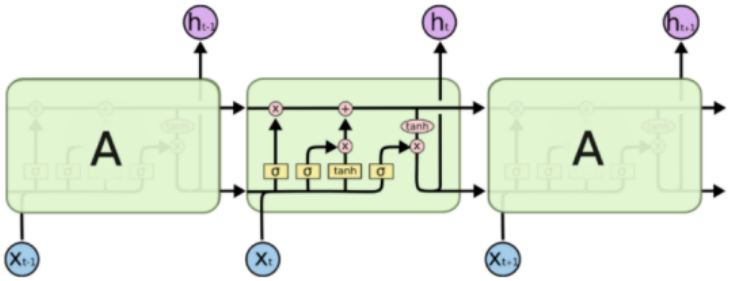


Figure 2.9. *The repeating module in an LSTM. Source: colah.github.io*

$\mathbf{x}_t$, with inputs $\mathbf{h}_{t-1}$ and $\mathbf{c}_{t-1}, \mathbf{h}_t$ and $\mathbf{c}_t$ are computed as follows:

$$\begin{aligned}
\mathbf{f}_t &= \sigma\left(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f\right) \\
\mathbf{i}_t &= \sigma\left(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o\right) \\
\mathbf{u}_t &= \tanh\left(\mathbf{W}_u \mathbf{x}_t + \mathbf{U}_u \mathbf{h}_{t-1} + \mathbf{b}_u\right) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{u}_t \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right)
\end{aligned}$$

Forget gate (f_t)**.** This gate decides what information should be thrown away or kept. Information from the previous hidden state h_{t-1} and information from the current input x_t is passed through the sigmoid activation function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

Input gate (fi_t)**.** The previous hidden state and current input are passed into a sigmoid function that decides which values will be updated by forcing the values to be between 0 and 1 (0 means unimportant and 1 means important). The hidden state and current input are also passed to the tanh function to squish values between -1 and 1 (u_t). Finally, the tanh output is multiplied with the sigmoid output $(i_t \odot u_t)$. The sigmoid output will filter the important information of tanh.

Cell state (c_t)**.** The cell state gets pointwise multiplied by the forget vector. This can drop values in the cell state if it gets multiplied by values near 0. Then, we take the output from the input gate and do a pointwise addition that updates the cell state to new values that the neural network finds relevant.

Output gate (o_t)**.** The output gate decides what the next hidden state should be. As the hidden state contains information on previous inputs, it is also used for predictions. First, the previous hidden state and the current input are passed into a sigmoid function. Then, the newly modified cell state is passed to the tanh function. We multiply the tanh output with the sigmoid output $(o_t \odot \tanh(\mathbf{c}_t))$ to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

2.5.3 Attention Models

Attention Model, first introduced for Machine Translation by Bahdanau et al. [52] and has now become a predominant concept in the neural network literature. Attention has become enormously popular within the Artificial Intelligence community as an essential component of neural architectures for many applications in Natural Language Processing, Speech, and Computer Vision.

The intuition behind attention can be best explained using human biological systems. For

example, our visual processing system tends to focus selectively on some parts of the image while ignoring other irrelevant information in a manner that can assist in perception [53]. Similarly, in several problems involving language, speech, or vision, some input parts are more critical than others. For instance, in machine translation and summarization tasks, only certain words in the input sequence may be relevant for predicting the next word. Likewise, in an image captioning problem, some input image regions may be more relevant for generating the next word in the caption. Attention Model incorporates this notion of relevance by allowing the model to dynamically pay attention to only certain parts of the input that help in performing the task at hand effectively.

The idea of attention can be understood using a regression model proposed by Naradaya-Watson in 1964 [54]. We are given training data of n instances comprising features and their corresponding target values. We want to predict the target value $\hat{y}$ or a new query instance x. A naive estimator will predict the simple average of target values of all training instances. Naradaya-Watson proposed a better approach in which the estimator uses a weighted average where weights correspond to the relevance of the training instance to the query. Indeed, the proposed an attention function $a(x, x_i)$ which is a weighting function that encodes the relevance of instance x_i predict for x. A common weighting function is a normalized Gaussian kernel, though other similarity measures can also be used with normalization. The authors showed that the estimator has (i) consistency: given enough training data, it converges to optimal results, and (ii) simplicity: no free parameters, the information is in the data and not in the weights. Fast forward 50 years, the attention mechanism in deep models can be viewed as a generalization that allows learning the weighting function.

The first use of Attention Model was proposed by Bahdanau et al. [52] for a sequence-to-sequence modeling task. A sequence-to-sequence model consists of an encoder-decoder architecture proposed by Cho et al. [55] as shown in Figure 2.10(a). The encoder is an RNN that takes an input sequence of tokens $\{x_1, x_2, \ldots, x_T\}$ where T is the length of input sequence, and encodes it into fixed length vectors $\{h_1, h_2, \ldots, h_T\}$. The decoder is also an RNN which then takes a single fixed length vector h_T as its input and generates an output sequence $\{y_1, y_2, \ldots, y_{T'}\}$ token by token, where T' is the length of output sequence. At each position t, h_t and s_t denote the hidden states of the encoder and decoder respectively.

There are two well-known challenges with this traditional encoder-decoder framework. First, the encoder compresses all the input information into a single fixed-length vector h_T that is passed to the decoder. Using a single fixed-length vector to compress long and detailed input sequences may lead to loss of information [55]. Second, it cannot model alignment between input and output sequences, which is an essential aspect of structured output tasks such as translation or summarization. Intuitively, in sequence-to-sequence tasks, each output token is expected to be more influenced by some specific parts of the input sequence. However, the decoder lacks any mechanism to selectively focus on relevant input tokens while generating each output token.

Attention Model aims at mitigating these challenges by allowing the decoder to access the entire encoded input sequence $\{h_1, h_2, \ldots, h_T\}$. The central idea is to induce attention weights α over the input sequence to prioritize the positions where relevant information is present for generating the next output token.

The corresponding encoder-decoder architecture with attention is shown in Figure 2.10(b). The attention block in the architecture is responsible for automatically learning the attention weights α_{ij}, which capture the relevance between h_i (the encoder hidden state) and s_{j-1} (the decoder hidden state). Note that the query state s_{j-1} is a hidden state of the decoder just before emitting s_j and y_j. These attention weights are then used for building a context vector c, which is passed as an input to the decoder. At each decoding position j, the context vector c_j is a weighted sum of all hidden states of the encoder and their corresponding attention weights, i.e. $c_j = \sum_{i=1}^{T} \alpha_{ij} h_i$. This additional context vector is the mechanism by which the decoder can access the entire input

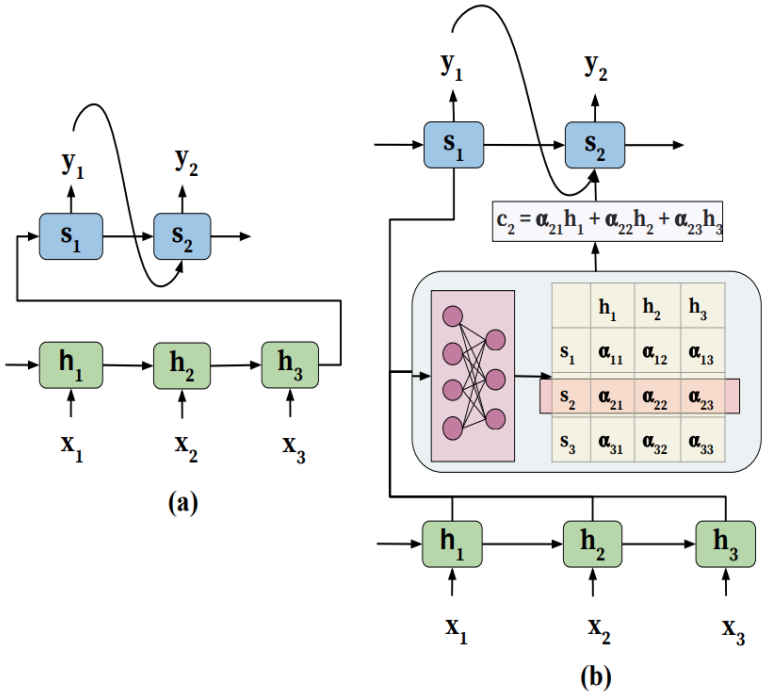


Figure 2.10. *Encoder-decoder architecture: (a) traditional (b) with attention model. Source: [5]*

sequence and focus on the relevant positions in the input sequence. This leads to improvements in performance on the final task and improves the quality of the output due to better alignment. The same concept is shown mathematically in Table 2.1. The only major difference in the encoder-decoder architecture with attention is the composition of context vector c. In the traditional framework, the context vector is just the last hidden state of the encoder h_T. In the attention-based framework, context at a given decoding step j is combination of all hidden states of the encoder and their corresponding attention weights: $c_j = \sum_{i=1}^{T} \alpha_{ij} h_i$.

The attention model shown in Figure 2.10(b) can also be seen as a mapping of a sequence of keys K to an attention distribution α according to query q where keys are encoder hidden states h_i and query is the single decoder hidden state s_{j-1}. Here the attention distribution α_{ij} emphasizes the keys which are relevant for the main task concerning the query q. Then $e = a(K, q)$ and $\alpha = p(e)$. In some cases, there is also additional input of values V on which the attention distribution is applied. The keys and values generally have one to one mapping, and although the core attention model proposed by Bahdanau et al. does not distinguish between keys and values ($k_i = v_i = h_i$), some existing literature uses this terminology for different representations of the same input data. Hence a generalized attention model A works with a set of key-value pairs (K, V) and query q such that:

$$A(q, K, V) = \sum_i p\left(a\left(k_i, q\right)\right) * v_i$$

Here the instance x is the query, the training data points x_i are keys, and their labels y_i are values. The alignment function (denoted by a) and distribution function (denoted by p) determine how keys and queries are combined to produce attention weights.

Alignment functions. The first major category of alignment functions is based on a notion of comparing query representations with key representations. For example, one approach is to compute either the cosine similarity or the dot product between the key and query representations. To account for varying lengths of representation, scaled dot product can be employed that normalizes the dot product by the representation vector length. Note that these functions assume that key

Function	Traditional Encoder-Decoder	Encoder-Decoder with Attention
Encode	$h_i = f(x_i, h_{i-1})$	$h_i = f(x_i, h_{i-1})$
Context	$c = h_T$	$c_j = \sum_{i=1}^{T} \alpha_{ij} h_i$
		$\alpha_{ij} = p(e_{ij})$
		$e_{ij} = a(s_{j-1}, h_i)$
Decode	$s_j = f(s_{j-1}, y_{j-1}, c)$	$s_j = f(s_{j-1}, y_{j-1}, c_j)$
Generate	$y_j = g(y_{j-1}, s_j, c)$	$y_j = g(y_{j-1}, s_j, c_j)$

Table 2.1. *Encoder-decoder architecture: traditional and with attention model.*
Notation: $x = (x_1, \ldots, x_T)$: input sequence, T: length of input sequence, h_i: hidden states of encoder, c: context vector, α_{ij}: attention weights over input, s_j: decoder hidden state,y_j: output token, f, g: non-linear functions, a: alignment function, p: distribution function

Function	Equation
similarity	$a(k_i, q) = \text{sim}(k_i, q)$
dot product	$a(k_i, q) = q^T k_i$
scaled dot product	$a(k_i, q) = \frac{q^T k_i}{\sqrt{d_k}}$
general	$a(k_i, q) = q^T W k_i$
biased general	$a(k_i, q) = k_i(Wq + b)$
activated general	$a(k_i, q) = act\left(q^T W k_i + b\right)$
generalized kernel	$a(k_i, q) = \phi(q)^T \phi(k_i)$
concat	$a(k_i, q) = w_{\text{imp}}^T \text{ act } (W[q; k_i] + b)$
additive	$a(k_i, q) = w_{\text{imp}}^T act\,(W_1 q + W_2 k_i + b)$
deep	$a(k_i, q) = w_{\text{imp}}^T E^{(L-1)} + b^L$
	$E^{(l)} = \text{act}\left(W_l E^{(l-1)} + b^l\right.$
	$E^{(1)} = \text{act}\,(W_1 k_i + W_0 q) + b^l$
location-based	$a(k_i, q) = a(q)$
feature-based	$a(k_i, q) = w_{\text{imp}}^T act\,(W_1 \phi_1(K) + W_2 \phi_2(K) + b)$

Table 2.2. *Summary of Alignment Functions.*
Notation: $a(k_i, q)$: alignment function for query q and key k_i, sim*: similarity functions such as cosine, d_k: length of input, $(W, w_{\text{imp}}, W_0, W_1, W_2)$: trainable parameters, b: trainable bias term,* act*: activation function.*

and query have the same representation vector space. General alignment extends dot product to keys and queries with different representations by introducing a learnable transformation matrix W that maps queries to the vector space of keys. Biased general alignment allows learning the global importance of some keys irrespective of the query by introducing a bias term. Activated general alignment adds a nonlinear activation layer such as hyperbolic tangent, rectifier linear unit, or scaled exponential linear unit. The formulations of these alignment functions are presented in the Table 2.2.

The second major category of alignment functions combines keys and queries to form a joint representation. One of the simplest models that follow this approach is the concat alignment by Luong et al.[56], where a joint representation is given by concatenating keys and queries. Additive alignment reduces computational time by decoupling the contributions of the query and the key; this allows precomputing contributions of all keys to avoid re-computation for each query. In contrast to a single neural layer used in additive alignment, deep alignment employs multiple neural layers.

2.5.4 Transformers

Transformer proposed by Vaswani et al. [7] is a prominent deep learning model that has been widely adopted in various fields, such as natural language processing (NLP), computer vision

(CV), and speech processing. Transformer was originally proposed as a sequence-to-sequence model for machine translation. However, later works show that Transformer-based pre-trained models (PTMs) can achieve state-of-the-art performances on various tasks. Consequently, Transformer has become the go-to architecture in NLP, especially for PTMs. In addition to language-related applications, Transformer has also been adopted in CV, audio processing, and even other disciplines, such as chemistry and life sciences.

The vanilla Transformer [7] is a sequence-to-sequence model and consists of an encoder and a decoder, each of which is a stack of L identical blocks. Each encoder block comprises a multi-head self-attention module and a position-wise feed-forward network (FFN). For building a deeper model, a residual connection [57] is employed around each module, followed by Layer Normalization [58] module. Compared to the encoder blocks, decoder blocks additionally insert cross-attention modules between the multi-head self-attention modules and the position-wise FFNs. Furthermore, the self-attention modules in the decoder are adapted to prevent each position from attending to subsequent positions. The overall architecture of the vanilla Transformer is shown in Figure 2.11. In the following subsection, we shall introduce the critical modules of the vanilla Transformer.

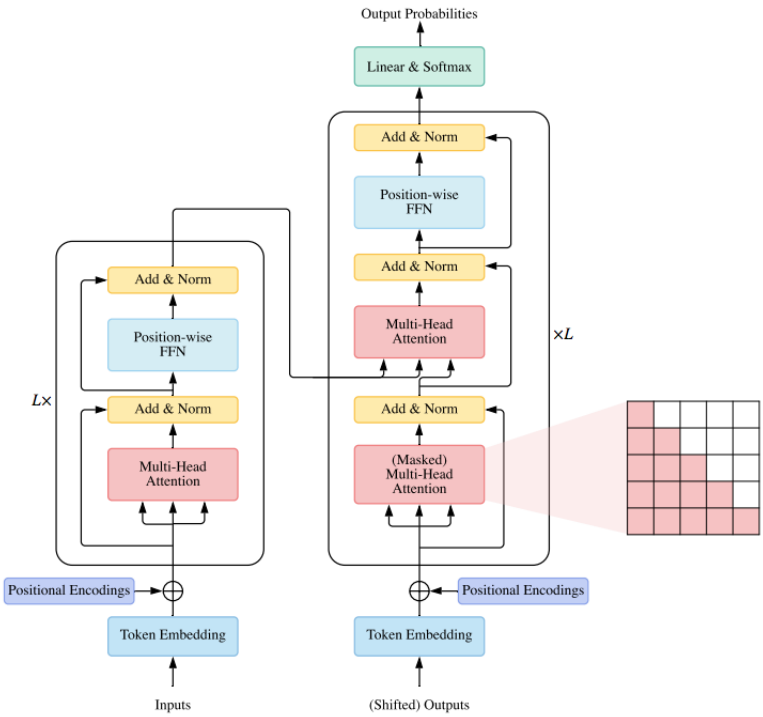


Figure 2.11. *Overview of vanilla Transformer architecture. Source: [6]*

Attention Modules. Transformer adopts attention mechanism with Query-Key-Value (QKV) model. Given the packed matrix representations of queries $Q \in \mathbb{R}^{N \times D_k}$, keys $K \in \mathbb{R}^{M \times D_k}$, and values $\mathbf{V} \in \mathbb{R}^{M \times D_v}$, the scaled dot-product attention used by Transformer is given by:

$$\text{Attention}\ (Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{D_k}}\right)\mathbf{V}$$

where N and M denote the lengths of queries and keys (or values); D_k and D_v denote the dimensions of keys (or queries) and values; $\mathbf{A} = \text{softmax}\left(\frac{\mathrm{QK}^\top}{\sqrt{D_k}}\right)$ is often called attention matrix; softmax is applied in a row-wise manner. The dot-products of queries and keys are divided by $\sqrt{D_k}$ to alleviate gradient vanishing problem of the softmax function.

Instead of simply applying a single attention function, Transformer uses multi-head attention,

where the D_m-dimensional original queries, keys, and values are projected into D_k, D_k and D_v dimensions, respectively, with H different sets of learned projections. For each of the projected queries, keys and values, and output is computed with attention. The model then concatenates all the outputs and projects them back to a D_m-dimensional representation. This can be illustrated in the Figure 2.12

$$\begin{aligned}\text{MultiHeadAttn}\left(\mathbf{Q},\mathbf{K},\mathbf{V}\right) &= \text{Concat}\left(\text{head}_1,\cdots,\text{head}_H\right)\mathbf{W}^O \\ \text{where head}_{\,i} &= \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q,\mathbf{K}\mathbf{W}_i^K,\mathbf{V}\mathbf{W}_i^V\right)\end{aligned}$$

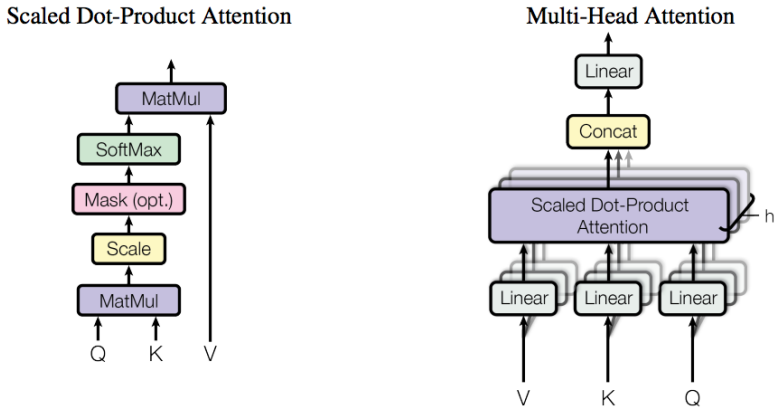


Figure 2.12. *(left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. Source: [7]*

In Transformer, there are three types of attention in terms of the source of queries and key-value pairs:

1. **Self-attention.** In Transformer encoder, we set $Q = K = V = X$, where X is the outputs of the previous layer.

2. **Masked Self-attention.** In the Transformer decoder, the self-attention is restricted such that queries at each position can only attend to all key-value pairs up to and including that position. To enable parallel training, this is typically done by applying a mask function to the unnormalized attention matrix $\hat{A} = \exp\left(\frac{QK^\dagger}{\sqrt{D_k}}\right)$, where the illegal positions are masked out by setting $\hat{A}_{ij} = -\infty$ if $i < j$. This kind of self-attention is often referred to as autoregressive or causal attention.

3. **Cross-attention.** The queries are projected from the outputs of the previous (decoder) layer, whereas the keys and values are projected using the outputs of the encoder.

Position-wise FFN. The position-wise FFN is a fully connected feed-forward module that operates separately and identically on each position

$$\text{FFN}\left(\text{H}'\right) = \text{ReLU}\left(\text{H}'\mathbf{W}^1 + \mathbf{b}^1\right)\mathbf{W}^2 + \mathbf{b}^2$$

where $\mathbf{H}'$ is the outputs of previous layer, and $\mathbf{W}^1 \in \mathbb{R}^{D_m \times D_f}, \mathbf{W}^2 \in \mathbb{R}^{D_f \times D_m}, \mathbf{b}^1 \in \mathbb{R}^{D_f}, \mathbf{b}^2 \in \mathbb{R}^{D_m}$ are trainable parameters. Typically the intermediate dimension D_f of the FFN is set to be larger than D_m.

Residual Connection and Normalization. In order to build a deep model, Transformer employs a residual connection [49] around each module, followed by Layer Normalization [4]. For

Module	Complexity	Parameters
self-attention	$O\left(T^2 \cdot D\right)$	$4D^2$
position-wise FFN	$O\left(T \cdot D^2\right)$	$8D^2$

Table 2.3. *Complexity and parameter counts of self-attention and position-wise FFN*

instance, each Transformer encoder block may be written as

$$\begin{aligned} \mathbf{H}' = & \text{ LayerNorm}(\text{ Selfattention }(\mathbf{X}) + \mathbf{X}) \\ \mathbf{H} = & \text{ LayerNorm }(\text{FFN}(\mathbf{H}') + \mathbf{H}') \end{aligned}$$

where SelfAttention $(\cdot)$ denotes self attention module and LayerNorm $(\cdot)$ denotes the layer normalization operation.

Position Encodings. Since Transformer does not introduce recurrence or convolution, and it is ignorant of positional information (especially for the encoder). Thus additional positional representation is needed to model the ordering of tokens. The authors employ the following equations to compute the positional encoding:

$$\begin{aligned} PE_{(\text{pos},2i)} &= \sin\left(\text{pos}/10000^{(2i/d_{\text{model}})}\right) \\ PE_{(\text{pos},2i+1)} &= \cos\left(\text{pos}/10000^{(2i/d_{\text{model}})}\right) \end{aligned}$$

Where pos is the position of the word in the sequence and i is the dimension. This positional encoding is added to the input embedding.

To illustrate the computation time and parameter requirements of the Transformer, we analyze the two core components of the Transformer (i.e., the self-attention module and the position-wise FFN) in Table 2.3. We assume that the hidden dimension D_m of the model is D, and that the input sequence length is T. The intermediate dimension of FFN is set to $4D$ and the dimension of keys and values are set to D/H.

When the input sequences are short, the hidden dimension D dominates the complexity of self-attention and position-wise FFN. The bottleneck of Transformer thus lies in FFN. However, as the input sequences grow longer, the sequence length T gradually dominates the complexity of these modules, in which case self-attention becomes the bottleneck of Transformer. Furthermore, the computation of self-attention requires that a $T \times T$ attention distribution matrix is stored, which makes the computation of Transformer infeasible for long-sequence scenarios (e.g., long text documents and pixel-level modeling of high-resolution images). One shall see that the goal of increasing the efficiency of Transformer generally leads to the long-sequence compatibility of self-attention and the computation and parameter efficiency of position-wise FFN for ordinary settings.

Analysis of Self-Attention. As a central piece of Transformer, self-attention comes with a flexible mechanism to deal with variable-length inputs. It can be understood as a fully connected layer where the weights are dynamically generated from pairwise relations from inputs. Table 2 compares the complexity, sequential operations, and maximum path length [4] of self-attention with three commonly used layer types. We summarize the advantages of self-attention as follows:

1. It has the same maximum path length as fully connected layers, making it suitable for long-range dependencies modeling. Compared to fully connected layers, it is more parameter efficient and more flexible in handling variable-length inputs.

2. Due to the limited receptive field of convolutional layers, one typically needs to stack a deep network to have a global receptive field. On the other hand, the constant maximum path

Layer Type	Complexity per Layer	Sequential Operations
Self-Attention	$O\left(T^2 \cdot D\right)$	$O(1)$
Fully Connected	$O\left(T^2 \cdot D^2\right)$	$O(1)$
Convolutional	$O\left(K \cdot T \cdot D^2\right)$	$O(1)$
Recurrent	$O\left(T \cdot D^2\right)$	$O(T)$

Table 2.4. *Per-layer complexity, minimum number of sequential operations and maximum path lengths for different layer types. T is the sequence length, D is the representation dimension and K is the kernel size of convolutions*

length enables self-attention to model long-range dependencies with a constant number of layers.

3. The constant sequential operations and maximum path length make self-attention more parallelizable and better at long-range modeling than recurrent layers.

Chapter 3

Natural Language Processing

3.1 Introduction

Natural Languages have evolved naturally in humans through use and repetition without conscious planning or premeditation, for example, English, Greek, and Latin. Therefore, they are distinguished from constructed and formal languages such as programming and logic modeling languages.

Natural Language Processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and natural language. Perhaps this definition is outdated because of the application of NLP in programming languages source code modeling and generation [59].

What is so special about natural language? Natural Language is a discrete/symbolic/categorical system specifically constructed to convey meaning and is not produced by a physical manifestation of any kind. In that way, it is very different from vision or any other machine learning task, where the input data, for example, the image, follow some underlying physical law. Most words are just symbols for an extra-linguistic entity: the word is a signifier that maps to a signified (idea or things).

The hierarchical levels of language that NLP examines are:

Phonology. This level deals with the interpretation of speech sounds within and across words. There are, in fact, three types of rules used in the phonological analysis: 1) phonetic rules – for sounds within words; 2) phonemic rules – for variations of pronunciation when words are spoken together, and; 3) prosodic rules – for fluctuation in stress and intonation across a sentence.

Morphology. This level deals with the componential nature of words composed of morphemes, the smallest units of meaning. Lexical. At this level, humans, as well as NLP systems, interpret the meaning of individual words. Several types of processing contribute to word-level understanding, the first of these being assignment of a single part-of-speech tag to each word. In this processing, words that can function as more than one part-of-speech are assigned the most probable part-of-speech tag based on the context in which they occur.

Syntactic. This level focuses on analyzing the words in a sentence to uncover the grammatical structure of the sentence.

Semantic. Semantic processing determines the possible meanings by focusing on the interactions among word-level meanings in the sentence. This level of processing can include the semantic disambiguation of words with multiple senses, in an analogous way to how syntactic disambiguation of words can function as multiple parts-of-speech is accomplished at the syntactic level. Semantic disambiguation permits one and only one sense of polysemous words to be selected and included in the semantic representation of the sentence.

Discourse. While syntax and semantics work with sentence-length units, the discourse level of NLP works with units of text longer than a sentence. It does not interpret multi-sentence texts

as just concatenated sentences, each of which can be interpreted singly. Rather, discourse focuses on the properties of the text that convey meaning by making connections between component sentences.

Pragmatic. This level is concerned with the purposeful use of language in various situations. The goal is to explain how extra meaning is read into texts without being encoded in text. This level requires world knowledge, including the understanding of intentions, plans, and goals.

3.2 Applications

Natural language processing provides both theory and implementations for a range of applications. Any application that utilizes text is a candidate for NLP. The most common applications of NLP include the following:

Information Retrieval and Web Search: the science of searching for documents, for information within documents, and metadata about documents, as well as searching databases and the World Wide Web.

Information Extraction (IE): the recognition, tagging, and extraction into a structured representation, certain critical elements of information, e.g., persons, companies, locations, organizations, from extensive collections of text.

Text Summarization: the process of distilling the essential information from a source in order to produce an abridged version.

Question Answering (QA): the response from documents to extracted or generated answer.

Machine Translation (MT): the use of computer software in order to translate text or speech from one natural language to another.

Speech Recognition and Synthesis: the extraction of a textual representation of a spoken utterance.

Text Generation: A method for generating sentences from "keywords".

Natural Language Understanding and Generation (NLU, NLG): NLG system is like a translator that converts a computer-based representation into a natural language representation.

Natural Language Inference (NLI) is the task of determining whether a "hypothesis" is true (entailment), false (contradiction), or undetermined (neutral) given a "premise".

3.3 Word Representation

The performance of any machine learning model is drastically dependent on input representation. In the field of NLP, the input is natural language and its components, words. Significant, in the English language, there are an estimated 13 million words, and thus there are 13 million possible inputs for a machine learning model. Ideally, the word representation should contain the word meaning, which is the idea represented by this word, and encode some properties such as a notion of similarity and difference between words and successful distinction polysemous words, which are words with different meanings on different occasions.

There are two ways that Linguists think about meaning, one is through “Denotational Semantics” which is the concept of representing an idea as a symbol (a word or a one-hot vector), and the other is through “Distributional Semantics” which is the concept of representing the meaning of a word based on the context in which it usually appears.

3.3.1 Denotational Representation

In this category, the word is mapped into a symbol, either a scalar or a vector.

Vocabulary IDs

Let us assume that we have a vocabulary V in a given task, containing all the words in the training, development, and test dataset. The most naive approach is to match all the words of the vocabulary with a unique ID symbol. For example for the given vocabulary $V = \{'I', 'love', 'cats'\}$, we can define the following mapping:

$$w^{\mathrm{I}} = 1, w^{love} = 2, w^{cats} = 3$$

This approach does not encapsulate the word's meaning through the representation, and neither a notion of similarity and difference between words nor the ambiguity problem is solved. Moreover, this representation is computationally expensive as it uses $|V|$ different IDs, and potentially this $|V|$ is 13 million. Finally, this approach is not capable of any further improvement.

One-Hot-encoding

One-Hot-Encoding is equivalent to the Vocabulary IDs, where we represent every word with an $\mathbb{R}^{1\times 1}$ vector. The only difference is the dimensionality augmentation of the subspace to achieve further improvement through compression techniques. In One-Hot-Encoding, every word is represented as an $\mathbb{R}^{|V|\times 1}$ vector with all 0 and one 1 at the index of that word in the sorted English language. So, for example, word vectors in this type of encoding would appear as the following:

$$w^{\mathrm{I}} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, w^{love} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, w^{cats} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

We represent each word as a completely independent symbol. As we previously discussed, this word representation does not directly give us a notion of similarity and difference between words, nor the ambiguity problem is solved. For instance,

$$\left(w^{\mathrm{cat}}\right)^T w^{\mathrm{dogs}} = \left(w^{\mathrm{siamese}}\right)^T w^{\mathrm{cat}} = 0$$

The corresponding matrix that describes the method will be of $|V| \times |V|$ dimension and in the following form:

$$\begin{bmatrix} w^{\mathrm{I}} \\ w^{\mathrm{love}} \\ w^{\mathrm{cats}} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Motivated by the fact that this matrix is sparse, we can apply many compression techniques such as Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF), but the final result should also be classified as "Denotational Semantics".

3.3.2 Distributional Semantics

As the British Linguist J.R. Firth said, "You shall know a word by the company it keeps" the following techniques study words regarding their context. Denotational Semantics word vectors can be classified into two categories: sparse and dense word vectors. Dense word vectors are often named word embeddings.

Sparse word vectors

Term Frequency–Inverse Document Frequency (TF-IDF) Representation. Tf-idf is a statistical measure used to determine the significance of words t in documents d. The term document could be defined as a batch of 10 words regardless of punctuations. However, it is essential to follow this declaration through the application of tf-idf in the given corpora. The tf-idf value comprises two terms: term frequency (tf) and inverse document frequency (idf). The TF term denotes the following intuition: frequently occurring words are more important than less frequent words. On the other hand, idf term denotes the opposite intuition: words that occur too often are unimportant. Thus, the tf-idf product tries to solve the trade-off between the tf and the idf term.

More specifically, tf proposed by Luhn [60] is the frequency of the word in the document. Usually, we use the count as the tf:

$$\mathrm{tf}_{t,d} = \mathrm{count}(t, d)$$

or a more squash term:

$$\mathrm{tf}_{t,d} = \log_{10}(\mathrm{count}(t, d) + 1)$$

The idf proposed by Sparck Jones [61] is defined as N/df_t , where N is the total number of documents, and df_t is the number of documents in which term t appears. The fewer documents in which a term occurs, the higher this weight. The lowest weight of 1 is assigned to terms that occur in all the documents. Usually, we use a more squash term for idf:

$$\mathrm{df}_t = \log_{10}(N/\mathrm{df}_t)$$

The tf-idf weighted value $w_{t,d}$, and the corresponding word vector, is the product of this two terms:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

Using tf-idf we represent every word with an $\mathbb{R}^{|D|\times 1}$ vector, where $|D|$ is the number of different documents in the collection and in general case $|D| < |N|$. Thus, the word representation is smaller than One-Hot-Vector and should contain some information about the word meaning, but tf-idf does not give us directly neither a notion of similarity and difference between words nor the ambiguity problem is solved.

Pointwise Mutual Information (PMI) Representation. Pointwise mutual information proposed by Fano [62] is one of the essential concepts in NLP. It is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

$$I(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

The pointwise mutual information, proposed by Church and Hanks [63], between a target word w and a context word c, where context can be the hole Vocabulary V, is then defined as:

$$\mathrm{PMI}(w, c) = \log_2 \frac{P(w, c)}{P(w)P(c)}$$

Because pointwise mutual information ranges from negative to positive infinity and the negative values are adverse, the proposed Positive PMI (PPMI). PPMI replaces negative PMI values with zero, and it can be defined as:

$$\text{PPMI}(w, c) = \max\left(\log_2 \frac{P(w, c)}{P(w)P(c)}, 0\right)$$

The numerator tells us how often we observed the two words together. The denominator tells us how often we would expect the two words to co-occur, assuming they occurred independently. Thus, the ratio estimates how much more the two words co-occur than we expect them to co-occur by chance.

More particularly, assuming we have $|W|$ words and $|C|$ contexts, then $f_{i,j}$ gives the number of times word w_i occurs in context c_j. This can be turned into a PPMI matrix where $\text{ppmi}_{\text{i,j}}$ gives the PPMI value of word w_i with context c_j and the word representations p_{i*} as follows:

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{|W|}\sum_{j=1}^{|C|} f_{ij}}, p_{i*} = \frac{\sum_{j=1}^{|C|} f_{ij}}{\sum_{i=1}^{|W|}\sum_{j=1}^{|C|} f_{ij}}, p_{*j} = \frac{\sum_{i=1}^{|W|} f_{ij}}{\sum_{i=1}^{|W|}\sum_{j=1}^{|C|} f_{ij}}, \text{PPMI}_{ij} = \max\left(\log_2 \frac{p_{ij}}{p_{i*}p_{*j}}, 0\right)$$

Using PMI we represent every word with an $\mathbb{R}^{|C|\times 1}$ vector, where $|C|$ is the number of contexts. PMI representation should contain some information about the word meaning but does not directly give us a notion of similarity and difference between words. Moreover, PMI does not overcome the ambiguity problem.

Dence word vectors

Sparse Vector representations are long vectors with a size of approximately around 50,000 in the average task. On the other hand, Dense Vectors, also called word embeddings and continuous vectors, have much smaller dimensionality, around 300-1200. On top of that, sparse representations are computationally expensive. Models trained with sparse vectors perform worse compared to the models based on dense vectors. On the other hand, representing words as 300-dimensional dense vectors requires our classifiers to learn far fewer weights than if we represented words as 50,000-dimensional vectors, and the smaller parameter space possibly helps with generalization and avoiding overfitting. Moreover, the dense representations generally achieve a better understanding of the meaning of the word.

Many different models were proposed for estimating continuous representations of words, including the well-known Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA).

Neural model architecture for estimating neural network language model (NNLM) was proposed in [9], where a feedforward neural network with a linear projection layer and a non-linear hidden layer was used to learn the word vector representation and a statistical language model jointly.

Another interesting architecture of NNLM was presented by Mikolov et al. [64], [65], where the word vectors are first learned using a neural network with a single hidden layer. The word vectors are then used to train the NNLM. Thus, the word vectors are learned even without constructing the full NNLM. In this study, we directly extend this architecture and focus on the first step, where the word vectors are learned using a simple model.

Finally, a Recurrent neural network-based language model has been proposed to overcome certain limitations of the feedforward NNLM, such as the need to specify the context length (the order of the model N), and because theoretically, RNNs can efficiently represent more complex patterns than the shallow neural networks [66]. Furthermore, the RNN model does not have a projection layer, only an input, hidden, and output layer.

The breakthrough methodology for dense word representation is Word2Vec proposed by Mikolov et al. [8]. Word2Vec is a shallow, two-layer neural network that is trained to reconstruct linguistic contexts of words. Given an input of a large corpus of words, it produces a vector space, typically of some hundred dimensions, with each word in the corpus being assigned a corresponding vector

in the space. Word vectors are located in the space such that words that share common contexts in the corpus are located close to one another in the space. Word2Vec has two forms, the continuous bag of words (CBOW) model and the Skip-Gram model as presented in Figure 3.1. When the feature vector assigned to a word cannot accurately predict that word's context, the components of the vector are adjusted. The vectors of words judged similar by their context are nudged closer together by adjusting the numbers in the vector.

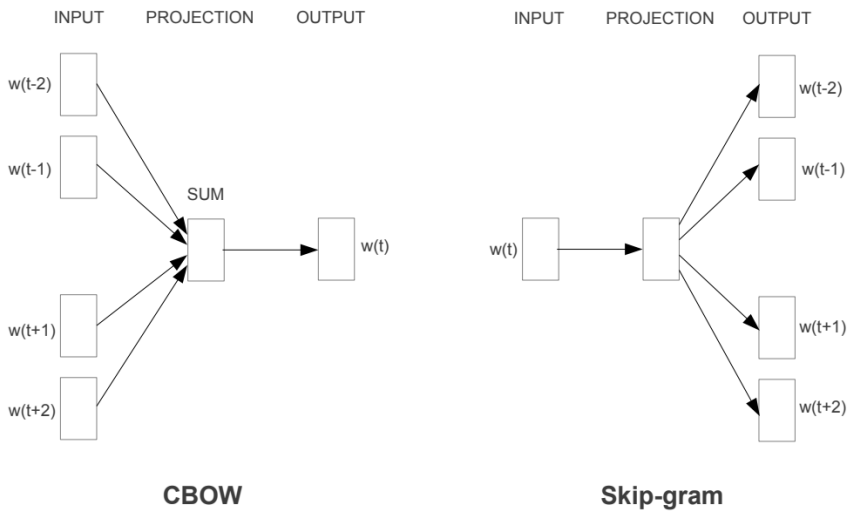


Figure 3.1. *The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. Source: [8]*

Word2vec embeddings encapsulate a notion of similarity and difference between words. However, they are static embeddings, meaning that the method learns one fixed embedding for each word, so it cannot overcome the problem of polysemous words. This problem can be overcome by learning dynamic contextual embeddings like BERT representations, in which the vector for each word is different in different contexts.

Word2Vec: Continuous Bag of Words (CBOW) The first proposed architecture of word2vec is CBOW which is like the feedforward NNLM, where the non-linear hidden layer is removed, and the projection layer is shared for all words (not just the projection matrix); thus, all words get projected into the same position (their vectors are averaged). We call this architecture a bag-of-words model as the order of words does not influence the projection. Furthermore, we also use words from the future; we have obtained the best performance on the task introduced in the next section by building a log-linear classifier with four future and four history words at the input, where the training criterion is to classify the current (middle) word correctly.

Word2Vec: Skip-Gram The second architecture of word2vec is like CBOW, but instead of predicting the current word based on the context, it tries to maximize the classification of a word based on another word in the same sentence. More precisely, we use each current word as an input to a log-linear classifier with a continuous projection layer and predict words within a specific range before and after the current word. We found that increasing the range improves the quality of the resulting word vectors, increasing the computational complexity. Furthermore, since the more distant words are usually less related to the current word than those close to it, we give less weight to the distant words by sampling less from those in our training examples.

GloVe Embeddings

GloVe, proposed by Pennington et al. [67], is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. GloVe is essentially a log-bilinear model with a weighted least-squares objective. The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. The central intuition underlying the model is the observation that ratios of word-word co-occurrence probabilities can encode some form of meaning. The GloVe model is trained on the non-zero entries of a global word-word co-occurrence matrix, which tabulates how frequently words co-occur with one another in a given corpus. Populating this matrix requires a single pass through the entire corpus to collect the statistics.

Contextual Embeddings

The distributional approaches aggregate the contexts in which a term occurs in a corpus. The result is a context-free, or else context-independent word representation. Word embeddings described so far are static. Nevertheless, usually, a particular word can be used in different sentences with a completely different meaning. Out of context, each word has multiple meanings. The obvious problem is that polysemous words (words with obvious multiple senses) cannot be modeled properly. For example, in the following sentences: "I dream of surfing the perfect wave." and "Will there be another wave of illness in the fall?", the word "wave" has a pretty different meaning. So static representations for words are quite insufficient solutions for understanding text. Therefore, it is proposed to represent words, depending on the context each time found. A famous algorithm for contextualized word representation is ELMo (Embeddings from Language Models) as proposed by Peters et al. /citepeters2018dee, and BERT (Bidirectional Encoder Representations from Transformers) by Jacob Devlin et al. [21].

3.4 Language Models

We want to develop a Language Model (LM) that can understand that the sentence: s_1 : "I love cats." is more likely to occur than the sentence: s_2 : "Cats book ice-cream". Indeed, the first sentence is entirely valid, syntactically and semantically; thus, a good language model will assign a higher probability to the first sentence.

More formally, Language Models compute the probability of occurrence of several words in a particular sequence. Mathematically, we can call this probability on any given sequence of n words:

$$P(w_1, w_2, \ldots, w_n)$$

This probability is equal to the following probability:

$$P\left(w_1, w_2, \ldots, w_n\right) = \prod_{i=1}^{n} P\left(w_i \mid w_{i-1}, w_{i-2}, \ldots, w_1\right) = P(w_1)P(w_2 \mid w_1)\ldots P(w_n \mid w_1, \ldots, w_{n-1})$$

3.4.1 Traditional Language Models

The formula for computing the probability of a sentence and thus describing a language model, called N-Gram LM, is computationally expensive. Thus, some assumptions are made for the efficient training of a language model.

The first and most naive assumption is the unary language model approach and breaks apart this probability by assuming the word occurrences are entirely independent. The LM is then a unigram, and the following formula can express the probability:

$$P(w_1, w_2, \cdots, w_n) = \prod_{i=1}^{n} P(w_i)$$

In order to train such a language model given a set of corpora C of $|N|$ total words, we have to calculate the following probability for each word w_i in the training corpora:

$$P(w_i) = \frac{\text{count}(w_i)}{|N|}, \forall w_i \in C$$

where $\text{count}(w_i)$ is the total time where the word w_i detected in the training corpora. The unigram Language Model is a bit ludicrous because the next word is contingent upon the previous words.

For a better LM we have to look back at Bayesian probability theory, and most precisely on the Markov condition, sometimes called the Markov assumption, which is an assumption made in Bayesian probability theory, that every node in a Bayesian network is conditionally independent of its non-descendants, given its parents. A more general assumption is the Causal Markov (CM) condition, which states that conditional on all its direct causes, a node is independent of all variables that are not direct causes or direct effects of that node. By using these theorems, we can construct Bigram LM, Trigram LM, and more complex LM.

A Bigram Language Model can formally be expressed as:

$$P(w_1, w_2, \cdots, w_n) = \prod_{i=2}^{n} P(w_i \mid w_{i-1})$$

In order to train a Bigram language model given a set of corpora C of $|N|$ total words, we have to calculate the following probability for each word w_i in the training corpora:

$$P(w_i \mid w_{i-1}) = \frac{\text{count}(w_i, w_{i-1})}{\sum_{w \in C} \text{count}(w_{i-1}, w)}, \forall w_i \in C$$

Finally, a Trigram Language Model can formally be expressed as:

$$P(w_1, w_2, \cdots, w_n) = \prod_{i=2}^{n} P(w_i \mid w_{i-1}, w_{i-2})$$

In order to train a Trigram language model given a set of corpora C of $|N|$ total words, we have to calculate the following probability for each word w_i in the training corpora:

$$P(w_i \mid w_{i-1}, w_{i-2}) = \frac{\text{count}(w_i, w_{i-1}, w_{i-2})}{\sum_{w \in C} \text{count}(w_{i-2}, w_{i-1}, w)}, \forall w_i \in C$$

Typically, the actual number of words in the history is based on how much training data are available. Trigram language models are commonly used, which consider a two-word history to predict the third word. Language models may also be estimated over 2-grams (bigrams), single words (unigrams), or any other order of n-grams.

3.4.2 Neural Language Models

Non-linear neural network models allow conditioning on increasingly large context sizes with only a linear increase in the number of parameters. For example, a neural probabilistic language model was popularized by Bengio et al. [9].

This model takes as an input vector representations of a word window of n previous words, looked up in a table C. These vectors are now known as word embeddings. These word embeddings, $C(w) \in \mathbb{R}^{d_w}$, are then concatenated and fed into a hidden layer, whose output is provided to a softmax layer as shown in Figure 3.2. This neural language model can be mathematically described as follow:

$$\mathbf{x} = [\mathbf{C}(w_1); \mathbf{C}(w_2); \ldots; \mathbf{C}(w_n)]$$
$$\hat{y} = P(w_i \mid w_{1:k}) = LM(w_{1:k}) = \text{softmax}(\mathbf{h}\mathbf{W}_2 + \mathbf{b}_2)$$
$$\mathbf{h} = g(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)$$
$$\mathbf{x} = [\mathbf{C}(w_1); \mathbf{C}(w_2); \ldots; \mathbf{C}(w_n)]$$
$$\mathbf{C}(w) = \mathbf{E}_{[w]}$$

where $w_i \in \mathcal{V}, \mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times d_w}, \mathbf{W}_1 \in \mathbb{R}^{n \cdot d_w \times d_{\text{hid}}}, \mathbf{b}_1 \in \mathbb{R}^{d_{\text{hid}}}, \mathbf{W}_2 \in \mathbb{R}^{d_{\text{hid}} \times |\mathcal{V}|}, \mathbf{b}_2 \in \mathbb{R}^{|\mathcal{V}|}$.

V is a finite vocabulary. The vocabulary size $|V|$, ranges between 1,000 - 1,000,000 words, with the common size being around 70,000 unique words. Feedforward neural networks have been replaced with recurrent neural networks [66] for language modeling.

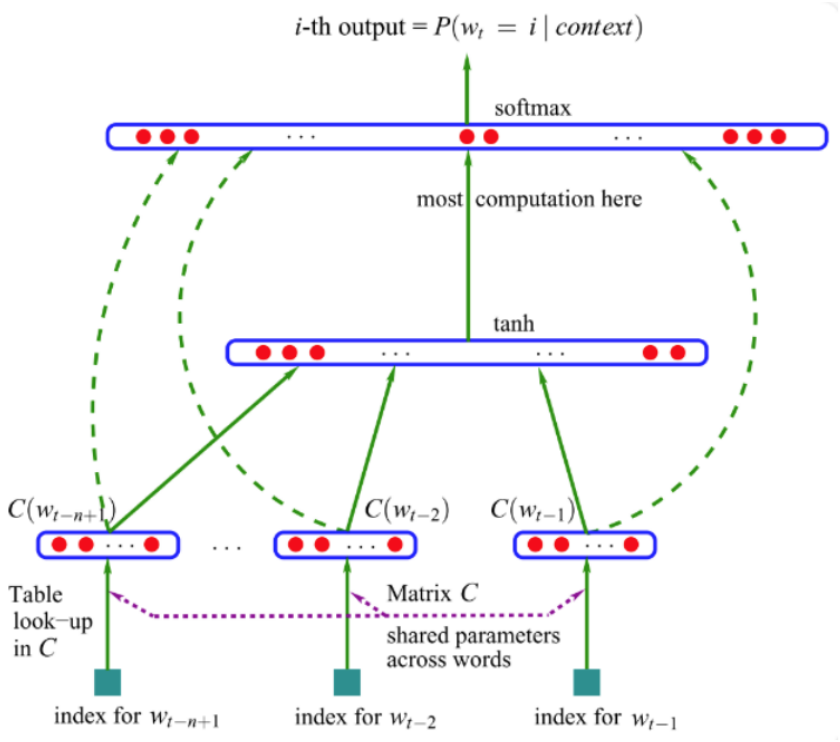


Figure 3.2. *A feed-forward neural network language model. Source: [9]*

3.5 Embeddings from Language Models (ELMo)

ELMo embeddings [68] are deep contextualized word representations that offer high-quality representations for language, modeling both complex characteristics of word use and adjusting them in different linguistic contexts. The vectors are derived from a bi-directional LSTM trained with a coupled language model (LM) objective on a large text corpus. ELMo representations are a function of all the internal layers of the bi-directional language model. However, the weighting of the ELMo embeddings needs to be carefully tuned for every different task. Therefore, the proposed method, although very effective, has some limitations and requires task-specific architectures.

3.6 Bidirectional Encoder Representations from Transformers (BERT)

BERT [10] proposed by J.Devlin et al. is a novel approach to incorporate bidirectionality in a single Transformer model. A particularly challenging task, direct approaches to incorporating bidirectionality in Transformer models fail since direct bidirectional conditioning would allow the words to see themselves in the light of context from multiple layers, thereby ruling out the possibility of using it as a Language Model. In essence, it was traditionally only possible to train a unidirectional encoder- a left-right or a right-left model. However, bidirectional models that could see the complete sequence context would inherently be more powerful than unidirectional models or a concatenation of two unidirectional models-left-right and right-left. To this end, the authors trained their model on two unsupervised prediction tasks:

Masked Language Model. To overcome the challenges posed while applying bi-directionality in Transformers, J.Devlin proposed masking of random tokens in the sequence. The Transformer was trained such that it had to predict only the words that had been masked while being able to view the whole sequence. WordPiece Tokenization is used to generate the sequence of tokens where rare words are split into sub-tokens. Then, masking of 15% of the Wordpiece Tokens is performed. Masking replaces the words with [MASK] tokens. However, instead of constantly replacing the selected words with a [MASK] token, the data generator employs the following approach:

- Replace the word with [MASK] token 80% of the time
- Replace the word with another random word 10% of time
- Keep the word as it is 10% of the time

Performing prediction on only 15% of all words instead of performing prediction on all words would entail that BERT would be much slower to converge. However, BERT showed immediate improvements in absolute accuracy while converging slightly slower than traditional unidirectional left-right models.

Next Sentence Prediction. This task entails predicting whether the first sequence provided immediately precedes the next. This task allows the Transformer to perform better on several downstream tasks such as question-answering, Natural Language Inference that involve understanding the relationship between two input sequences. The dataset used for training had a balanced 50/50 distribution created as follows: choosing an actual pair of neighboring sentences for positive examples and a random choice of the second sentence for the negative examples. The input sequence for this pair classification task is generated as:

$$[\text{CLS}] < \text{SentenceA} > [\text{SEP}] < \text{SentenceB} > [\text{SEP}]$$

Where sentences A and B are two sentences after performing the masking operations. The [CLS] token is the first token used to obtain a fixed vector representation that is consequently used for classification, and [SEP] is used to separate the two input sequences. As a result, the authors were able to achieve an impressive accuracy of $97 - 98\%$ in the next sentence prediction task.

Pre-Training Procedure. The authors have used the BooksCorpus and the English Wikipedia as pretraining data. They have used two variations of BERT-BASE(12- layer) and BERT-LARGE(24 layers)- that primarily differ in their depth. The maximum length of the input sequence is restricted to 512 tokens. All subsequent tokens in the sequence are neglected. A dropout value of 0.1 is used as regularization. Furthermore, the authors have made use of the GELU instead of Relu as an activation function. GELU- Gaussian Error Linear units have been shown to provide improvements compared to ReLU and eLu. Training of the models was performed on TPUs. Specifically, BERT-BASE was trained on 16 TPU chips for four days. BERT-LARGE was trained on 64 TPU chips, also for four days.

Fine-Tuning Procedure. The pre-trained BERT can be finetuned on a relatively small dataset and requires lesser processing power, as it can be described in the Figure 3.3. BERT improved upon the previous state-of-the-art in several tasks involving natural language inference, question answering, semantic similarity, linguistic acceptability, among other tasks. The pattern of the input and output sequence varies depending on the type of the task. The tasks can be broadly divided into four categories:

- Single Sentence Classification Tasks: These tasks are performed by adding layers on the classification embedding [CLS] and passing the input sequence preceded by the [CLS] token.
- Sentence Pair Classification Tasks The two sentences are passed to BERT after being separated by the [SEP] token. Classification can be performed by adding layers to the [CLS]
- Question Answering Tasks
- Single Sentence Tagging Tasks

Subsequently, two multilingual BERT models-uncased and cased-for over 102 languages were released. Furthermore, OpenAI released the GPT2 [69], essentially BERT trained as a language model on a considerable amount of data.

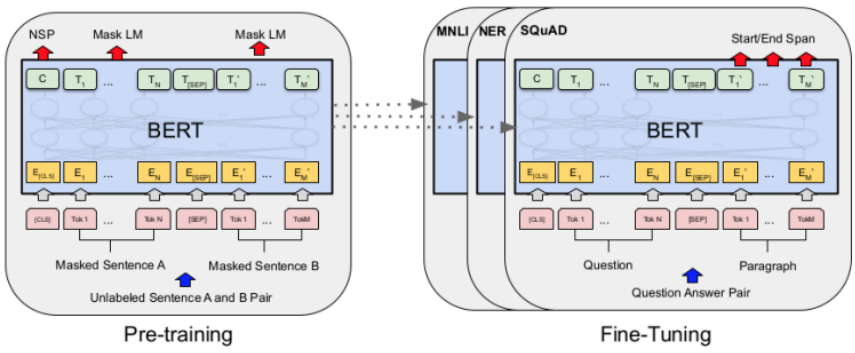


Figure 3.3. *Pre-training and fine-tuning procedures for BERT. Source: [10]*

Mathematical Notation Attention is a core component of Transformers, which consist of several layers, each containing multiple attentions ("heads"). Each attention head gathers relevant information from the input vectors. A vector is updated by vector transformations, attention

weights, and a summation of vectors.Mathematically, attention computes each output vector $\boldsymbol{y}_i \in \mathbb{R}^d$ from the corresponding pre-update vector $\widetilde{\boldsymbol{y}}_i \in \mathbb{R}^d$ and a sequence of input vectors $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subseteq \mathbb{R}^d$:

$$\boldsymbol{y}_i = \left(\sum_{j=1}^{n} \alpha_{i,j} \boldsymbol{v}\left(\boldsymbol{x}_j\right) \right) \boldsymbol{W}^O$$

$$\alpha_{i,j} := \mathrm{softmax}_{\boldsymbol{x}_j \in \mathcal{X}} \left(\frac{\boldsymbol{q}\left(\widetilde{\boldsymbol{y}}_i\right) \boldsymbol{k}\left(\boldsymbol{x}_j\right)^\top}{\sqrt{d'}} \right) \in \mathbb{R}$$

where $\alpha_{i,j}$ is the attention weight assigned to the token x_j for computing y_i, and $\boldsymbol{q}(\cdot), \boldsymbol{k}(\cdot)$, and $\boldsymbol{v}(\cdot)$ are the query, key, and value transformations, respectively.

$$\boldsymbol{q}\left(\widetilde{\boldsymbol{y}}_i\right) := \widetilde{\boldsymbol{y}}_i \boldsymbol{W}^Q + \boldsymbol{b}^Q \quad \left(\boldsymbol{W}^Q \in \mathbb{R}^{d \times d'}, \boldsymbol{b}^Q \in \mathbb{R}^{d'}\right)$$

$$\boldsymbol{k}\left(\boldsymbol{x}_j\right) := \boldsymbol{x}_j \boldsymbol{W}^K + \boldsymbol{b}^K \left(\boldsymbol{W}^K \in \mathbb{R}^{d \times d'}, \boldsymbol{b}^K \in \mathbb{R}^{d'}\right)$$

$$\boldsymbol{v}\left(\boldsymbol{x}_j\right) := \boldsymbol{x}_j \boldsymbol{W}^V + \boldsymbol{b}^V \left(\boldsymbol{W}^V \in \mathbb{R}^{d \times d'}, \boldsymbol{b}^V \in \mathbb{R}^{d'}\right)$$

Attention gathers value vectors $\boldsymbol{v}\left(\boldsymbol{x}_j\right)$ based on attention weights and then, applies matrix multiplication $\boldsymbol{W}^O \in \mathbb{R}^{d' \times d}$. Boldface letters such as $\boldsymbol{x}$ denote row (not column) vectors, following the notations in Vaswani et al.. In self-attention, the input vectors $\mathcal{X}$ and the pre-update vector $\widetilde{\boldsymbol{y}}_i$ are previous layer's output representations. In source-target attention, $\mathcal{X}$ corresponds to the representations of the encoder, and vector $\widetilde{\boldsymbol{y}}_i$ (and updated vector $\boldsymbol{y}_i$) corresponds to the vector of the i-th input token of the decoder.

3.7 GLUE Benchmark

For natural language understanding (NLU) technology to be maximally useful, it must be able to process language in a way that is not exclusive to a single task, genre, or dataset. In pursuit of this objective, we examine the General Language Understanding Evaluation (GLUE) benchmark [11], a collection of tools for evaluating the performance of models across a diverse set of existing NLU tasks. GLUE benchmark contains the following Task and datasets:

1. Single-Sentence Task
   (a) CoLA: Corpus of Linguistic Acceptability
   (b) SST-2: Stanford Sentiment Treebank
2. Similarity and Paraphrase Tasks
   (a) MRPC: Microsoft Research Paraphrase Corpus
   (b) QQP: Quora Question Pairs
   (c) STS-B: Semantic Textual Similarity Benchmark
3. Inference Tasks
   (a) MNLI: Multi-Genre Natural Language Inference
   (b) QNLI: Question-answering Natural Language Inference
   (c) RTE: Recognizing Textual Entailment
   (d) WNLI: Winograd Natural Language Inference

In Figure 3.4 there is an overall description of the GLUE Datasets.

Corpus	|Train|	|Test|	Task	Metrics	Domain
			Single-Sentence Tasks		
CoLA	8.5k	**1k**	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
			Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	**391k**	paraphrase	acc./F1	social QA questions
			Inference Tasks		
MNLI	393k	**20k**	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	**146**	coreference/NLI	acc.	fiction books

Figure 3.4. *Task descriptions and statistics. All tasks are single sentence or sentence pair classification, except STS-B, which is a regression task. MNLI has three classes; all other classification tasks have two. Test sets shown in bold use labels that have never been made public in any form. Source: [11]*

3.7.1 CoLA

The Corpus of Linguistic Acceptability [70] consists of English acceptability judgments drawn from books and journal articles on linguistic theory. Each example is a sequence of words annotated with whether it is a grammatical English sentence. Matthew's correlation coefficient is used for evaluation metric, which evaluates performance on unbalanced binary classification and ranges from -1 to 1, with 0 being the performance of uninformed guessing. GLUE uses the standard test set, for which we obtained private labels from the authors. GLUE reports a single performance number on the combination of the in and out-of-domain sections of the test set.

3.7.2 SST-2

The Stanford Sentiment Treebank [71] consists of sentences from movie reviews and human annotations of their sentiment. The task is to predict the sentiment of a given sentence. GLUE uses the two-way (positive/negative) class split and uses only sentence-level labels.

3.7.3 MRPC

The Microsoft Research Paraphrase Corpus [72] is a corpus of sentence pairs automatically extracted from online news sources, with human annotations for whether the sentences in the pair are semantically equivalent. Because the classes are imbalanced (68 percent positive), GLUE follows common practice and reports both accuracy and F1 score.

3.7.4 QQP

The Quora Question Pairs dataset is a collection of question pairs from the community question-answering website Quora. The task is to determine whether a pair of questions are semantically equivalent. As in MRPC, the class distribution in QQP is unbalanced (63 percent negative), so GLUE reports both accuracy and F1 score. Again, GLUE uses the standard test set, for which we obtained private labels from the authors.

3.7.5 STS-B

The Semantic Textual Similarity Benchmark [73] is a collection of sentence pairs drawn from news headlines, video and image captions, and natural language inference data. Each pair is human annotated with a similarity score from 1 to 5; the task predicts these scores. Following standard practice, GLUE evaluates using Pearson and Spearman correlation coefficients.

3.7.6 MNLI

The Multi-Genre Natural Language Inference Corpus [74] is a crowdsourced collection of sentence pairs with textual entailment annotations. Given a premise sentence and a hypothesis sentence, the task is to predict whether the premise entails the hypothesis (entailment), contradicts the hypothesis (contradiction), or neither (neutral). The premise sentences are gathered from ten sources, including transcribed speech, fiction, and government reports. GLUE uses the standard test set, for which private labels were obtained from the authors and evaluated on both the matched (in-domain) and mismatched (cross-domain) sections.

3.7.7 QNLI

The Stanford Question Answering Natural Language Inference Dataset [75] is a question-answering dataset consisting of question-paragraph pairs, where one of the sentences in the paragraph (drawn from Wikipedia) contains the answer to the corresponding question (written by an annotator). GLUE converts the task into sentence pair classification by forming a pair between each question and each sentence in the corresponding context and filtering out pairs with low lexical overlap between the question and the context sentence. The task is to determine whether the context sentence contains the answer to the question. This modified version of the original task removes the requirement that the model selects the exact answer and removes the simplifying assumptions that the answer is always present in the input and that lexical overlap is reliable.

3.7.8 RTE

The Recognizing Textual Entailment datasets come from a series of annual textual entailment challenges. Examples are constructed based on news and Wikipedia text. GLUE converts all datasets to a two-class split.

3.7.9 WNLI

The Winograd Natural Language Inference Schema Challenge [76] is a reading comprehension task in which a system must read a sentence with a pronoun and select the referent of that pronoun from a list of choices. The examples are manually constructed to foil simple statistical methods: Each one is contingent on contextual information provided by a single word or phrase in the sentence. GLUE constructs sentence pairs by replacing the ambiguous pronoun with each possible referent to convert the problem into sentence pair classification. The task is to predict if the original sentence entails the sentence with the pronoun substituted. GLUE uses a small evaluation set consisting of new examples derived from fiction books5 that were shared privately by the authors of the original corpus. While the included training set is balanced between two classes, the test set is imbalanced (65 percent not entailment). Also, due to a data quirk, the development set is adversarial: hypotheses are sometimes shared between training and development examples, so if a model memorizes the training examples, they will predict the wrong label on the corresponding development set example. Finally, as with QNLI, each example is evaluated separately, so there

is not a systematic correspondence between a model's score on this task and its score on the unconverted original task.

3.7.10 SciERC

SciERC [24] includes annotations for scientific entities, their relations, and coreference clusters for 500 scientific abstracts. These abstracts are taken from 12 AI conference/workshop proceedings in four AI communities from the Semantic Scholar Corpus. These conferences include general AI (AAAI, IJCAI), NLP (ACL, EMNLP, IJCNLP), speech (ICASSP, Interspeech), machine learning (NIPS, ICML), and computer vision (CVPR, ICCV, ECCV).

3.7.11 PubMed 200k RCT

PubMed 200k RCT [25] is a dataset based on PubMed for sequential sentence classification. The dataset consists of approximately 200,000 abstracts of randomized controlled trials, totaling 2.3 million sentences. Each abstract sentence is labeled with its role in the abstract using one of the following classes: background, objective, method, result, or conclusion.

Chapter 4

Compression of Deep Learning Models

4.1 Introduction

Transformer-based models keep growing by orders of magnitude: The 110M parameters of base BERT are now dwarfed by 17B parameters of Turing-NLG [22], which is dwarfed by 175B of GPT-3 [23]. This trend raises concerns about computational complexity of self-attention, environmental issues [34] [35], fair comparison of architectures [36], and reproducibility.

Human language is incredibly complex and would perhaps take many more parameters to describe fully, but the current models do not make good use of the parameters they already have. For example, Voita et al. [14] showed that all but a few Transformer heads could be pruned without significant losses in performance. For BERT, Clark et al. [27] observe that most heads in the same layer show similar self-attention patterns (perhaps related to the fact that the output of all self-attention heads in a layer is passed through the same MLP), which explains why Michel et al. [1] were able to reduce most layers to a single head.

Depending on the task, some BERT heads/layers are not only redundant [38], but also harmful to the downstream task performance. Positive effect from head disabling was reported for machine translation [1], abstractive summarization [39], and GLUE tasks [13]. Additionally, Tenney et al. [40] examine the cumulative gains of their structural probing classifier, observing that in 5 out of 8 probing tasks, some layers cause a drop in scores (typically in the final layers). From the aspect of unstructured pruning, Gordon et al. [28] find that 30–40 percent of the weights can be pruned without impact on downstream tasks.

In general, larger BERT models perform better [41] , [42] but not always: BERT-base outperformed BERT-large on subject-verb agreement [43] and sentence subject detection [44]. Given the complexity of language and amounts of pre-training data, it is not clear why BERT ends up with redundant heads and layers. Clark et al. [27] suggest that one possible reason is the use of attention dropouts, which causes some attention weights to be zeroed-out during training.

Given the above evidence of **overparameterization**, it does not come as a surprise that BERT can be efficiently compressed with minimal accuracy loss, which would be highly desirable for real-world applications.

4.2 Compression: Problem Setting

The goal of model compression is to achieve a smaller model from the original model without significantly reducing the accuracy. The simplified model is a model that is reduced in size and latency compared to the original model. Specifically, reducing size means that the compression model has fewer and smaller parameters and therefore uses less RAM at runtime, which is desirable because it frees up other parts of the memory application program. Latency reduction is the reduction in the time required for the model to make predictions or inferences based on the input

of the training model, which usually translates into a reduction in energy consumption during runtime. Model size and latency are usually closely related because larger models require more memory accesses to run. Some well-known research methodologies for compressing neural models are the following:

- Pruning
- Quantization
- Knowledge distillation
- Neural Architecture Search (NAS)

4.2.1 Pruning

Pruning involves removing connections between neurons or entire neurons, channels, or filters from a trained network, which is done by zeroing out values in its weights matrix or removing groups of weights entirely. For example, to prune a single connection from a network, one weight is set to zero in a weight's matrix, and to prune a neuron, all values of a column in a matrix are set to zero. The motivation behind pruning is that networks tend to be over-parametrized, with multiple features encoding nearly the same information. Pruning can be divided into two types based on the type of network component removed: unstructured pruning involves removing individual weights or neurons, while structured pruning involves removing entire channels or filters. We will look at these two types individually, as they differ in their implementations and outcomes.

Unstructured Pruning By replacing connections or neurons with zeros in a weight matrix, unstructured pruning increases the sparsity of the network, i.e., its proportion of zero to non-zero weights. Depending on the degree of sparsity and the method of storage used, pruned networks can also take up much less memory than their dense counterparts.

However, what are the criteria for deciding which weights should be removed? One standard method known as magnitude-based pruning compares the weights' magnitudes to a threshold value. A highly cited 2015 paper by Han et al. [77] prompted widespread adoption of this approach. In their implementation, pruning is applied layer-by-layer. First, a predetermined "quality parameter" is multiplied by the standard deviation of a layer's weights to calculate the threshold value, and weights with magnitudes below the threshold are zeroed. After all layers are pruned, the model is retrained so that the remaining weights can adjust to compensate for those removed, and the process is repeated for several iterations.

Mathematical formalism of Structured Pruning Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$, and a desired sparsity level κ (i.e., the number of non-zero weights) neural network weight pruning can be written as the following constrained optimization problem:

$$\min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}) = \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(\mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)\right)$$
$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m, \quad \|\mathbf{w}\|_0 \leq \kappa$$

Here, $\ell(\cdot)$ is the standard loss function (*e.g.*, cross-entropy loss), $\mathbf{w}$ is the set of parameters of the neural network, m is the total number of structural sets and $\|\cdot\|_0$ is the standard L_0 norm. There is no efficient way to minimize the L_0 norm as it is non-convex, NP-hard, and requires combinatorial search.

Structured Pruning Unlike unstructured pruning, structured pruning does not result in weight matrices with problematic sparse connectivity patterns because it involves removing entire blocks of weights within given weight matrices. Thus pruned model can be run using the same

hardware and software as the original. While we are now looking at groups of weights to remove at the channel or filter level, magnitude-based pruning can still be applied by ranking them according to their L_1 norms, for example. However, there are also more intelligent, "data-driven" approaches that have been proposed which can achieve better results.

Huang et al. [78] were the first to integrate into the pruning process a means of controlling the tradeoff between network performance and size. Their algorithm outputs a set of filters "pruning agents"—each a neural network corresponding to a convolutional layer of the network—and an alternative, pruned version of the original model, which is initialized to be the same as the original. The pruning agents maximize an objective that is parametrized by a "drop bound" value, defined as the maximum allowed drop in performance between the original and the pruned model, forcing the agents to keep performance above a specified level. A pruning agent is trained for each convolutional layer by evaluating the effects of pruning combinations of different filters within that layer. To do so, it removes certain filters from the alternative model and compares this model's performance on an evaluation set to that of the original, learning which modifications will increase the network's efficiency while still adhering to accuracy constraints. Once the agent for one layer is trained, and filters for that layer have been optimally removed, the entire pruned model is retrained to adjust for the changes, and the process repeats for the next convolutional layer.

Mathematical formalism of Unstructured Pruning Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, and a desired sparsity level κ (i.e., the number of non-zero structural sets) neural network structural pruning can be written as the following constrained optimization problem:

$$\min_{\mathbf{w_s}} L(\mathbf{w_s}; \mathcal{D}) = \min_{\mathbf{w_s}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(\mathbf{w_s}; (\mathbf{x}_i, \mathbf{y}_i)\right)$$
$$\text{s.t.} \quad \mathbf{w_s} \in \mathbb{R}^m, \quad \|\mathbf{w_s}\|_0 \leq \kappa$$

Here, $\ell(\cdot)$ is the standard loss function (*e.g.*, cross-entropy loss), $\mathbf{w_s}$ is the structural set of parameters of the neural network e.g. a convolutional filter, m is the total number of structural sets and $\|\cdot\|_0$ is the standard L_0 norm. There is no efficient way to minimize the L_0 norm as it is non-convex, NP-hard, and requires combinatorial search.

Accelerating unstructured sparse matrix multiplication is an active area of research in which recent progress has been made. For example, bank-balanced sparsity (which is closely related to unstructured sparsity) achieves near-ideal speed-ups while requiring a minimal deviation from unstructured sparsity [48]. On the systems side, adaptive sparse matrix multiplication has shown promising results on GPUs but is not applied in silicon.

4.2.2 Quantization

While pruning compresses models by reducing the number of weights, quantization consists of decreasing the weights' size. Quantization, in general, is the process of mapping values from a large set to values in a smaller set, meaning that the output consists of a smaller range of possible values than the input, ideally without losing too much information in the process. In a neural network, a particular layer's weights or activation outputs tend to be normally distributed within a specific range, so ideally, a quantization schema takes advantage of this fact and adapts to fit each layer's distribution. For example, models weights' are typically stored as 32-bit floating-point numbers, and a common approach is to reduce these to 8-bit fixed points, for which there are 256 possible values. We can see how quantizing the weights in this way would reduce their memory footprint by one-fourth.

4.3 Lottery Ticket Hypothesis (LTH)

The pruning pipeline contains the following sequential stages: firstly, train the entire initial network and then prune the trained network, based on a pruning algorithm.

Can we train the pruned network from the beginning and achieve similar or even better performance? The answer to this question is negative unless the untrained network has a good initialization. Frank et al. [12] proposed that the good initialization is the initial weights before the training-pruning pipeline.

The proposed pipeline is the following: train the initial model, apply a pruning algorithm to detect the pruned network, and then retrain the network by initializing the non-pruned connection weight equal to the corresponding weights of the initial model. More formally, this is known as Lottery Ticket Hypothesis, and it can be stated as: "A randomly-initialized, dense neural network contains a subnetwork that is initialized such that —when trained in isolation— it can match the test accuracy of the original network after training for at most the same number of iterations".

Frank et al. not only proposed a pruning pipeline but also showed why deep neural networks work. By overparameterization, many subnetworks are created inside the deep neural network from which some subnetworks can achieve competitive performance alone. The best of these subnetworks are called "Winning Ticket". The "Winning Tickets" generalize across vision datasets [79] and exist both in LSTM and Transformer models for NLP [80].

Identifying winning tickets. Frank et al. identify a winning ticket by training a network and pruning its smallest-magnitude weights, which is the simplest unstructured pruning methodology. The remaining unpruned connections constitute the architecture of the winning ticket. Each unpruned connection's value is then reset to its initialization from the original network before it was trained. The following pipeline can describe this process:

1. Randomly initialize a neural network $f(x;\theta_0)$ (where $\theta_0 \sim \mathcal{D}_\theta$).
2. Train the network for j iterations, arriving at parameters θ_j.
3. Prune $p\%$ of the parameters in θ_j, creating a mask m.
4. Reset the remaining parameters to their values in θ_0, creating the winning ticket $f(x; m \odot \theta_0)$.

As described, this pruning approach is one-shot: the network is trained once, $p\%$ of weights are pruned, and the surviving weights are reset. This process can be graphically described by Figure 4.1. However, Frank et al. focus on iterative pruning (IP), which repeatedly trains, prunes, and resets the network over n rounds; each round prunes $p^{\frac{1}{n}}\%$ of the weights that survive the previous round. Their results show that iterative pruning finds winning tickets that match the accuracy of the original network at smaller sizes than does one-shot pruning.

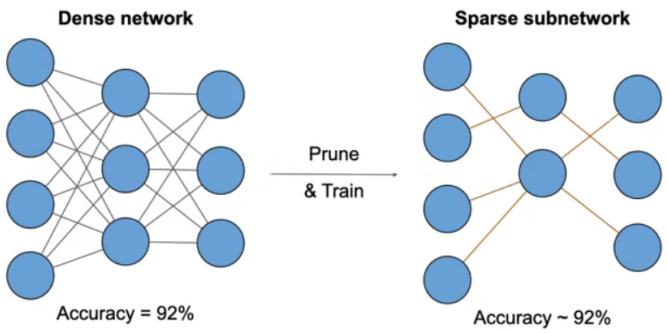


Figure 4.1. *Graphic Illustration of Lottery Ticket Hypothesis*

Figure 4.2 presents the results of the lottery ticket hypothesis as applied to a LeNet-300-100 architecture proposed by LeCun et al. [81] trained on MNIST. As shown in Figure 4.2 the final

performance of the pruned model for a large variety of pruning rates is greater than the performance of the entire network.

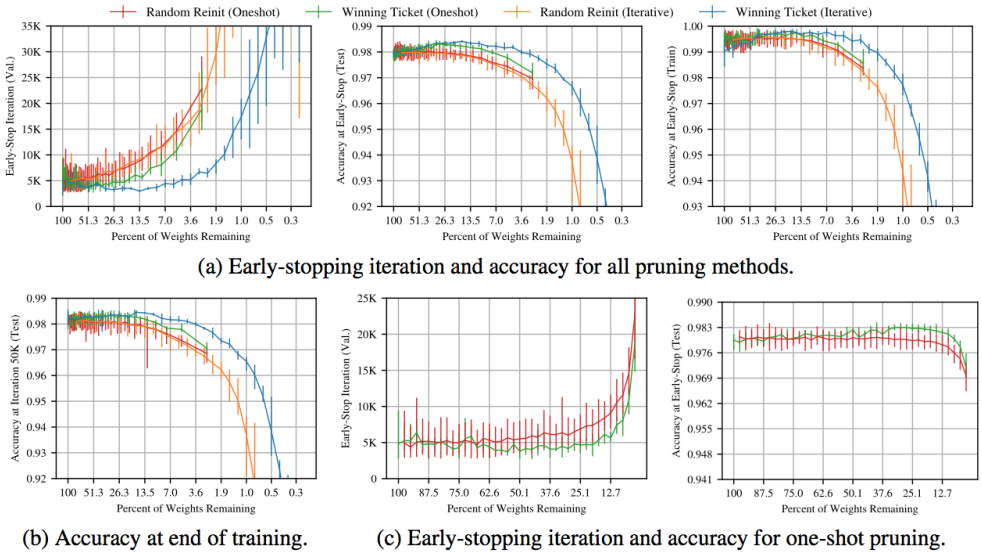


Figure 4.2. *Early-stopping iteration and accuracy of LeNet under one-shot and iterative pruning. Average of five trials; error bars for the minimum and maximum values. Source: [12]*

4.4 Pruning Transformer-based models

4.4.1 Transformer-based Structured Pruning

BERTology and Structured Pruning. BERTology [37] is a growing field of study concerned with investigating the inner working of large-scale transformers like BERT. Many pruning procedures based on BERTology results about the nature of attention heads.

Kovaleva et al.[13] suggest that there is a limited set of attention patterns that are repeated across different heads, indicating the overall model overparameterization. While different heads consistently use the same attention patterns, they have varying impacts on performance across different tasks. They show that manually disabling attention in certain heads improves performance over the regular fine-tuned BERT models.

For their study, the primary tool is self-attention maps: self-attention weights are extracted for each head in every layer for a given input. Self-attention map is a $2D$ float array of shape $L \times L$, where L is the length of an input sequence.

They detect the following BERT's self-attention pattern as it can be shown in the Figure 4.3:

- Vertical: mainly corresponds to attention to special BERT tokens [CLS] and [SEP], which serve as delimiters between individual chunks of BERT's inputs
- Diagonal: formed by the attention to the previous/following tokens
- Vertical and Diagonal: a mix of the previous two types,
- Block: intra-sentence attention for the tasks with two distinct sentences (such as, for example, RTE or MRPC),
- Heterogeneous: highly variable depending on the specific input and cannot be characterized by a distinct structure. Source: [13]

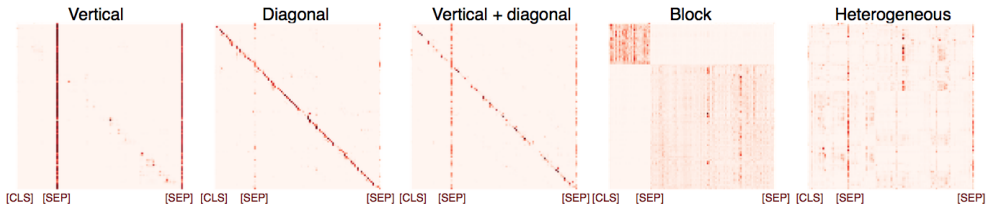


Figure 4.3. *Typical self-attention classes used for training a neural network. Both axes on every image represent BERT tokens of an input example, and colors denote absolute attention weights (darker colors stand for greater weights). The first three types are most likely associated with language model pre-training, while the last two potentially encode semantic and syntactic information. Source: [13]*

They conclude that the estimated upper bound on all heads in the "Heterogeneous" category varies from 32 (MRPC) to 61 percent (QQP), depending on the task.

Moreover, they studied the Relation-specific heads in BERT. For each sentence of the pre-processed FrameNet Dataset (Figure 4.4), proposed by Baker et al. [82], they obtain pre-trained BERT's attention weights for each of the 144 heads. Every head returns the maximum absolute attention weight among those token pairs that correspond to the annotated semantic link contained within a given sentence. Then they average the derived scores over all the collected examples. They conclude that 2 out of 144 heads tend to attend to the sentence parts that FrameNet annotators identified as core elements of the same frame.

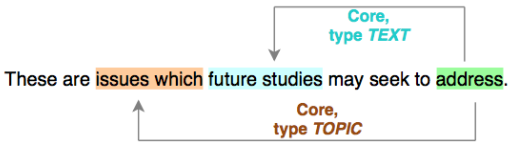


Figure 4.4. *FrameNet annotation example for the "address" lexical unit with two core frame elements of different types annotated. Source: [13]*

With this observes they tried to prune BERT, and since it relies heavily on the learned attention weights, they define head disabling as modifying the attention values of a head to be constant $a = 1/L$ for every token in the input sentence, where L is the length of the sentence. Thus, every token receives the same attention, effectively disabling the learned attention patterns while maintaining the information flow of the original model.

They conclude the following:

- Disabling some heads leads not to a drop in accuracy, as one would expect, but to an increase in performance.
- While disabling some heads improves the results, disabling the others hurts the results. However, it is important to note that disabling some heads across all tasks and datasets leads to an increase in performance.
- A random head gives, on average, an increase in performance. Furthermore, disabling a whole layer, that is, all 12 heads in a given layer, also improves the results.
- Relation-specific heads are 2 out of 144 heads, but they do not appear to be important in any GLUE tasks: disabling either one does not lead to a drop in accuracy. This implies that fine-tuned BERT does not rely on this piece of semantic information and prioritizes other features instead.

Voita et al. [14] study Transformer Model for the task of Neural Machine Translation and evaluate the contribution made by individual attention heads in the encoder to the overall performance of the model and analyze the roles played by them. They study heads using Layer-wise relevance propagation (LRP) [83]: a method for computing the relative contribution of neurons at one point in a network to neurons at another. They proposed to use LRP to evaluate the degree to which different heads at each layer contribute to the top-1 logit predicted by the model. Heads whose outputs have a higher relevance value may be judged to be more critical to the model's predictions.

They conclude that LRP ranks a small number of heads in each layer as much more important than all others. A graphic representation of the result is the following for a model trained on 6m OpenSubtitles EN-RU data can be found in the Figure 4.5

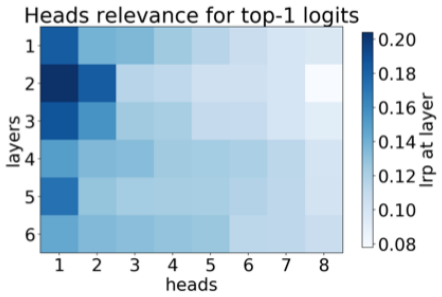


Figure 4.5. *Importance (according to LRP) of self-attention heads. The model trained on 6m OpenSubtitles EN-RU data. Source: [14]*

By examining some attention matrices paying particular attention to heads ranked highly by LRP, they categorize the heads in the following categories:

- Positional: the head points to an adjacent token,
- Syntactic: the head points to tokens in a specific syntactic relation
- Rare Words: the head points to the least frequent tokens in a sentence.

Structured Pruning by gating heads Another contribution of Voita et al. [14] was the proposed attention heads pruning methodology for the Transformer Architecture. In order to perform structured pruning, they modify the original Transformer architecture by multiplying the representation computed by each head $_i$ by a scalar gate g_i :

$$\text{MultiHead}(Q, K, V) = \text{ Concat }_i \left(g_i \cdot \text{ head }_i\right) W^O$$

Unlike usual gates, g_i are parameters specific to heads and are independent of the input (i.e., the sentence). The aim is to disable less critical heads entirely rather than simply downweighing them, thus would ideally apply L_0 regularization to the scalars g_i. The L_0 norm equals the number of non-zero components and would push the model to switch off less essential heads:

$$L_0\left(g_1, \ldots, g_h\right) = \sum_{i=1}^{h} \left(1 - [[g_i = 0]]\right)$$

h is the number of heads, and [[ ]] denotes the indicator function.

The L_0 norm is nondifferentiable and so cannot be directly incorporated as a regularization term in the objective function. Thus, they use a stochastic relaxation: each gate g_i is now a random

variable drawn independently from a head-specific distribution. They use the Hard Concrete distributions as proposed by Louizos et al. [84], a parameterized family of mixed discrete-continuous distributions over the closed interval $[0, 1]$.

The distributions have non-zero probability mass at 0 and 1, $P(g_i = 0 \mid \phi_i)$ and $P(g_i = 1 \mid \phi_i)$, where ϕ_i are the distribution parameters. Now the sum of the probabilities of heads being non-zero can be used as a relaxation of the L_0 norm:

$$L_C(\phi) = \sum_{i=1}^{h} (1 - P(g_i = 0 \mid \phi_i))$$

The new training objective is:

$$L(\theta, \phi) = L_{xent}(\theta, \phi) + \lambda L_C(\phi)$$

where θ are the parameters of the original Transformer, $L_{xent}(\theta, \phi)$ is cross-entropy loss for the translation model, and $L_C(\phi)$ is the regularizer described above.

To determine which head functions are most important in the encoder and how many heads the model needs, they conduct a series of experiments with gates applied only to encoder self-attention. The overall pipeline here is to prune a model by **fine-tuning a trained model with the regularized objective**.

Through Figure 4.6 and Figure 4.7 it is noticeable that most heads, especially the encoder self-attention heads, can be removed without seriously affecting performance.

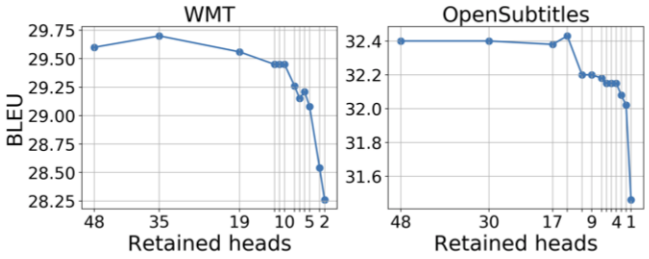


Figure 4.6. *BLEU score as a function of number of re-tained encoder heads (EN-RU). Regularization applied by fine-tuning trained model. Source: [14]*

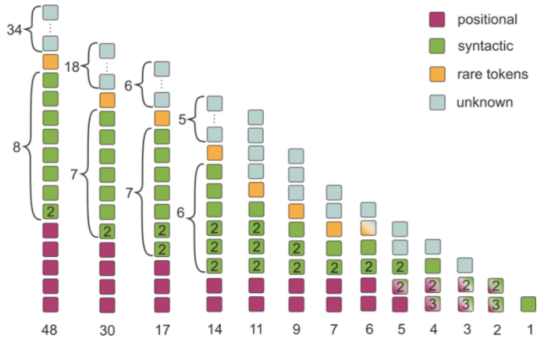


Figure 4.7. *Functions of encoder heads retained after pruning. Each column represents all remaining heads after varying amount of pruning (EN-RU; Subtitles). Source: [14]*

Michel et al. [1] perform ablation experiments on the attention heads by modifying the formula

for Multi-Head Attention:

$$\mathrm{MHAtt}(\mathbf{x}, q) = \sum_{h=1}^{N_h} \xi_h \mathrm{Att}_{W_k^h, W_q^h, W_v^h, W_o^h}(\mathbf{x}, q)$$

where the ξ_h are mask variables with values in $\{0, 1\}$.In order to mask head h, we simply set $\xi_h = 0$.

As a proxy score for head importance, Michel et al. look at the expected sensitivity of the model to the mask variables ξ_h defined above:

$$I_h = \mathbb{E}_{x \sim X} \left| \frac{\partial \mathcal{L}(x)}{\partial \xi_h} \right|$$

Where X is the data distribution and $\mathcal{L}(x)$ the loss on sample x. Intuitively, if I_h has a high value, then changing ξ_h is liable to have a significant effect on the model. They find the absolute value crucial to avoid data points with highly negative or positive contributions from nullifying each other in the sum. Plugging the above Equations and applying the chain rule yields the following final expression for I_h :

$$I_h = \mathbb{E}_{x \sim X} \left| \mathrm{Att}_h(x)^T \frac{\partial \mathcal{L}(x)}{\partial \, \mathrm{Att}_h(x)} \right|$$

As far as performance is concerned, estimating I_h only requires performing a forward and backward pass, and therefore is not slower than training. In practice, we compute the expectation over the training data or a subset thereof. Finally, the importance scores are normalized by layer (using the ℓ_2 norm).

Michel et al. concluded that if models have been trained using multiple heads, in practice, a large percentage of attention heads can be removed at test time without significantly impacting performance. Some layers can even be reduced to a single head.

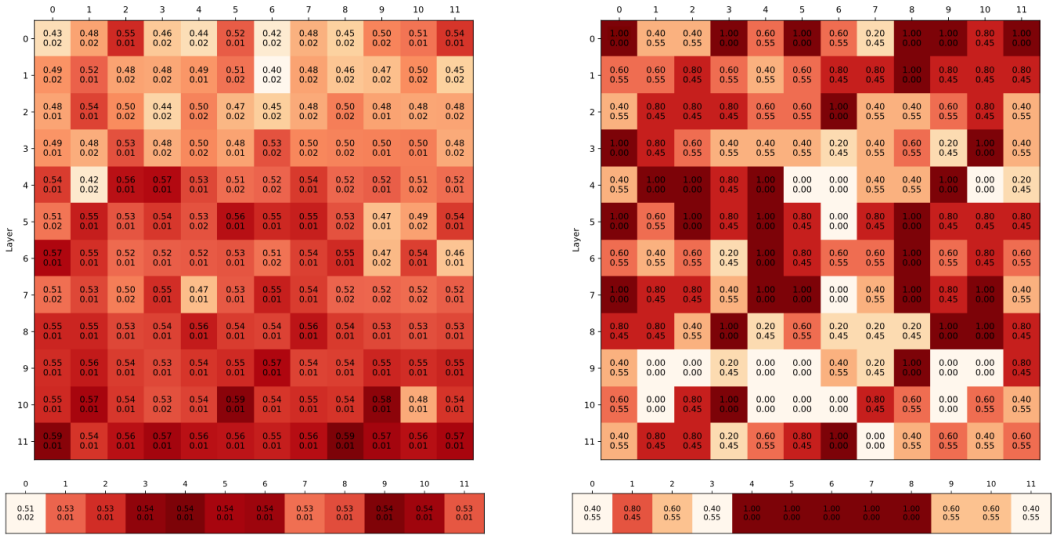
Prasanna et al. [15] apply magnitude and structural pruning on fine-tuned BERT and then re-finetune it. For the magnitude pruning approach, an iteratively pruning 10% of the lowest magnitude weights across the entire model is applied. Regarding the structured pruning approach, they use Michel et al. heuristic to determine whether a self-attention head is essential for the end task.

They compute head and MLP importance scores in a single backward pass, prune 10% heads and one MLP with the minor scores and continue the process until the performance on the development set is more significant than 90% performance of the full fine-tuned model. They found some patterns regarding the pruned heads, and these patterns for the QNLI task can be found in Figure 4.8:

Prasanna et al. perform structured lottery ticket hypothesis in different masks; classified as "good" and "bad" masks. Their results can be found in the Figure 4.9.

Regarding the multi-tasking learning aspect, Prasanna et al. [15] study the overlaps in BERT's "good" subnetworks self-attention heads between GLUE tasks. The overlaps in the "good" subnetworks are not explainable by two tasks' relying on the same linguistic patterns in individual self-attention heads. They also do not seem to depend on the type of task. For instance, consider that two tasks targeting paraphrases (MRPC and QQP) have less in common than MRPC and MNLI. Alternatively, the overlaps may indicate shared heuristics or patterns somehow encoded in combinations of BERT elements.

Their results can be found in Figure 4.10.



(a) M-pruning: each cell gives the percentage of surviving weights, and std across 5 random seeds.

(b) S-pruning: each cell gives the average number of random seeds in which a given head/MLP survived and std.

Figure 4.8. *The "good" subnetworks for QNLI: self-attention heads (top,* 12×12 *heatmaps) and MLPs (bottom,* 1×12 *heatmaps), pruned together. Earlier layers start at* 0*. The experiment ran with* 5 *random initializations and reported averages and standard deviations. Source: [15]*

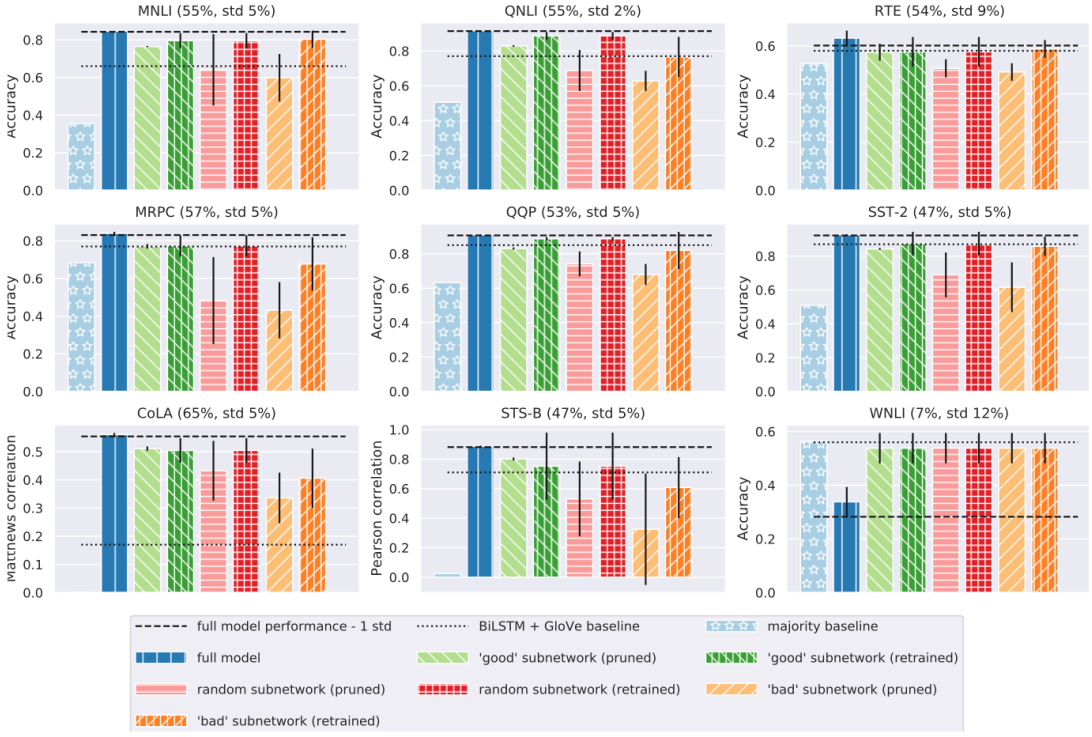


Figure 4.9. *The "good" and "bad" subnetworks in BERT fine-tuning: performance on GLUE tasks. 'Pruned' subnetworks are only pruned, and 'retrained' subnetworks are restored to pre-trained weights and fine-tuned. Subfigure titles indicate the task and percentage of surviving weights. STD values and error bars indicate standard deviation of surviving weights and performance, respectively, across* 5 *fine-tuning runs. Source: [15]*

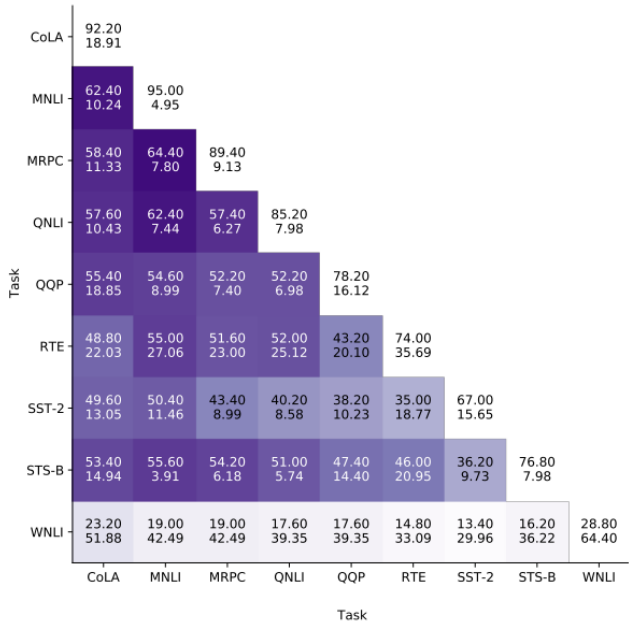


Figure 4.10. *Overlaps in BERT's "good" subnetworks be- tween GLUE tasks: self-attention heads. Source: [15]*

4.4.2 Transformer-based Magnitude Pruning

Lottery Ticket Hypothesis and Magnitude Pruning. Chen et al. [16] applied the unstructured Lottery Ticket Hypothesis on pretrained BERT. They study the accuracy when training subnetworks of neural networks. For a network $f(x;\theta,\cdot)$, a subnetwork is a network $f(x;m\odot\theta,\cdot)$ with a pruning mask $m\in\{0,1\}^{d_1}$ (where $\odot$ is the element-wise product). That is, it is a copy of $f(x;\theta,\cdot)$ with some weights fixed to 0.

Let $\mathcal{A}_t^{\mathcal{T}}\left(f\left(x;\theta_i,\gamma_i\right)\right)$ be a training algorithm (e.g., AdamW with hyperparameters) for a task $\mathcal{T}$ (e.g., CoLA) that trains a network $f\left(x;\theta_i,\gamma_i\right)$ on task $\mathcal{T}$ for t steps, creating network $f\left(x;\theta_{i+t},\gamma_{i+t}\right)$. Let θ_0 be the BERT-pre-trained weights. Let $\epsilon^{\mathcal{T}}(f(x;\theta))$ be the evaluation metric of model f on task $\mathcal{T}$.

A subnetwork $f(x;m\odot\theta,\gamma)$ is matching for an algorithm $\mathcal{A}_t^{\mathcal{T}}$ if training $f(x;m\odot\theta,\gamma)$ with algorithm $\mathcal{A}_t^{\mathcal{T}}$ results in evaluation metric on task $\mathcal{T}$ no lower than training $f\left(x;\theta_0,\gamma\right)$ with algorithm $\mathcal{A}_t^{\mathcal{T}}$. In other words:

$$\epsilon^{\mathcal{T}}\left(\mathcal{A}_t^{\mathcal{T}}\left(f(x;m\odot\theta,\gamma)\right)\right)\geq\epsilon^{\mathcal{T}}\left(\mathcal{A}_t^{\mathcal{T}}\left(f\left(x;\theta_0,\gamma\right)\right)\right)$$

A subnetwork $f(x;m\odot\theta,\gamma)$ is a winning ticket for an algorithm $\mathcal{A}_t^{\mathcal{T}}$ if it is a matching subnetwork for $\mathcal{A}_t^{\mathcal{T}}$ and $\theta=\theta_0$.

A subnetwork $f\left(x;m\odot\theta,\gamma_{\mathcal{T}_i}\right)$ is universal for tasks $\{\mathcal{T}_i\}_{i=1}^N$ if it is matching for each $\mathcal{A}_{t_i}^{\mathcal{T}_i}$ for appropriate, task-specific configurations of $\gamma_{\mathcal{T}_i}$.

To identify subnetworks $f(x;m\odot\theta,\cdot)$, they use neural network pruning [16, 17]. They determine the pruning mask m by training the unpruned network to completion on a task $\mathcal{T}$ (i.e., using $\mathcal{A}_t^{\mathcal{T}}$) and pruning individual weights with the lowest-magnitudes globally throughout the network. Since the goal is to identify a subnetwork for the pre-trained initialization of the state of the network early in training, they set the weights of this subnetwork to θ_i for a specific rewinding step i in training. For example, to set the weights of the subnetwork to their values from the pre-trained initialization, they set $\theta=\theta_0$. Previous work has shown that it is better to repeat this pruning process iteratively. The Iterative Magnitude Pruning (IMP) can be described in the following

algorithm 4.1.

ALGORITHM 4.1: *Iterative Magnitude Pruning (IMP) to sparsity s with rewinding step i.*

1: Train the pre-trained network $f(x; \theta_0, \gamma_0)$ to step $i : f(x; \theta_i, \gamma_i) = \mathcal{A}_i^{\mathcal{T}}(f(x; \theta_0, \gamma_0))$.
2: Set the initial pruning mask to $m = 1^{d_1}$.
3: **repeat**
4: Train $f(x; m \odot \theta_i, \gamma_i)$ to step $t : f(x; m \odot \theta_t, \gamma_t) = \mathcal{A}_{t-i}^{\mathcal{T}}(f(x; m \odot \theta_i, \gamma_i))$.
5: Prune 10% of remaining weights of $m \odot \theta_t$ and update m accordingly.
6: **until** the sparsity of m reaches s
7: **return** $f(x; m \odot \theta_i)$

The results described in Figure 4.11 suggest that although BERT is a pre-trained language model and thus is not randomly initialized, the Lottery Ticket Hypothesis can be implied through Iterative Magnitude Pruning.

Dataset	MNLI	QQP	STS-B	WNLI	QNLI	MRPC	RTE	SST-2	CoLA	SQuAD	MLM
Sparsity	70%	90%	50%	90%	70%	50%	60%	60%	50%	40%	70%
Full $\text{BERT}_{\text{BASE}}$	82.4 ± 0.5	90.2 ± 0.5	88.4 ± 0.3	54.9 ± 1.2	89.1 ± 1.0	85.2 ± 0.1	66.2 ± 3.6	92.1 ± 0.1	54.5 ± 0.4	88.1 ± 0.6	63.5 ± 0.1
$f(x, m_{\text{IMP}} \odot \theta_0)$	82.6 ± 0.2	90.0 ± 0.2	88.2 ± 0.2	54.9 ± 1.2	88.9 ± 0.4	84.9 ± 0.4	66.0 ± 2.4	91.9 ± 0.5	53.8 ± 0.9	87.7 ± 0.5	63.2 ± 0.3
$f(x, m_{\text{RP}} \odot \theta_0)$	67.5	76.3	21.0	53.5	61.9	69.6	56.0	83.1	9.6	31.8	32.3
$f(x, m_{\text{IMP}} \odot \theta_0')$	61.0	77.0	9.2	53.5	60.5	68.4	54.5	80.2	0.0	18.6	14.4
$f(x, m_{\text{IMP}} \odot \theta_0'')$	70.1	79.2	19.6	53.3	62.0	69.6	52.7	82.6	4.0	24.2	42.3

Figure 4.11. *Performance of subnetworks at the highest sparsity for which IMP finds winning tickets on each task. To account for fluctuations, we consider a subnetwork to be a winning ticket if its performance is within one standard deviation of the unpruned BERT model. Entries with errors are the average across five runs, and errors are the standard deviations. IMP = iterative magnitude pruning; RP = randomly pruning; θ_0 = the pre-trained weights; θ_0' = random weights; θ_0'' = randomly shuffled pre-trained weights. Source: [16]*

4.5 Pruning Computer Vision models

In the literature, idea exchange between NLP and Computer Vision (CV) is presented. Thus in this section, some ideas regarding pruning CV models are described.

Mallya et al. [17] introduce PackNet, which uses iteratively pruning for unimportant weights and fine-tuning them for learning new tasks. As a result of pruning and weight modifications, a binary parameter usage mask is produced by PackNet. PackNet algorithm can be described in Figure 4.12.

Another work, "Piggyback", proposed by Mallya et al. [18] does not change weights of the initial backbone network and learns a different mask per task. As a result, it is agnostic to task order, and the addition of a task does not affect performance on any other task. Further, an unlimited number of tasks can piggyback onto a backbone network by learning a new mask. Piggyback algorithm can be described in Figure 4.13.

4.6 Pruning Using Explainable AI (XAI) Techniques

For a machine learning model to generalize well, one must ensure that meaningful patterns in the input data support its decisions. However, a prerequisite is for the model to be able to explain itself, e.g., by highlighting which input features it uses to support its prediction. Layer-wise Relevance Propagation (LRP) is a technique that brings such explainability and scales to potentially highly complex deep neural networks.

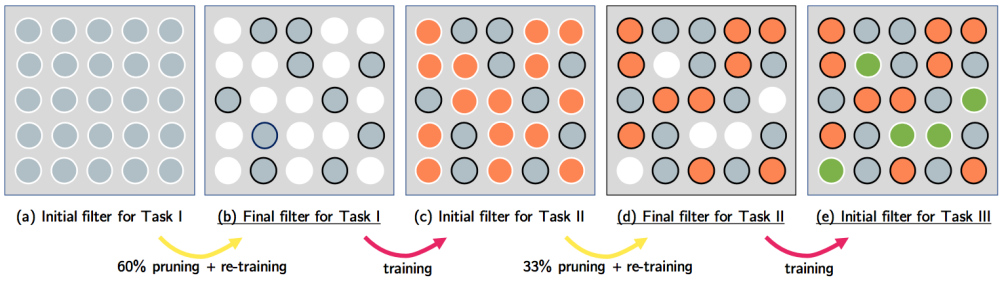


Figure 4.12. *Illustration of the evolution of a* 5×5 *filter with steps of training. Initial training of the network for Task I learns a dense filter as illustrated in (a). After pruning by* 60% *and re-training, a sparse filter for Task I is obtained, as depicted in (b), where white circles denote* 0 *valued weights. Weights retained for Task I are kept fixed for the remainder of the method and are not eligible for further pruning. The pruned weights are allowed to be updated for Task II, leading to filter (c), which shares weights learned for Task I. Another round of pruning by* 33% *and re-training leads to filter (d), the filter used for evaluating Task II (Note that weights for Task I, in gray, are not considered for pruning). Hereafter, weights for Task II, depicted in orange, are kept fixed. This process is completed until desired or runs out of pruned weights, as shown in the filter (e). The final filter (e) for Task III shares weights learned for tasks I and II. At test time, appropriate masks are applied depending on the selected Task to replicate filters learned for the respective tasks. Source: [17]*

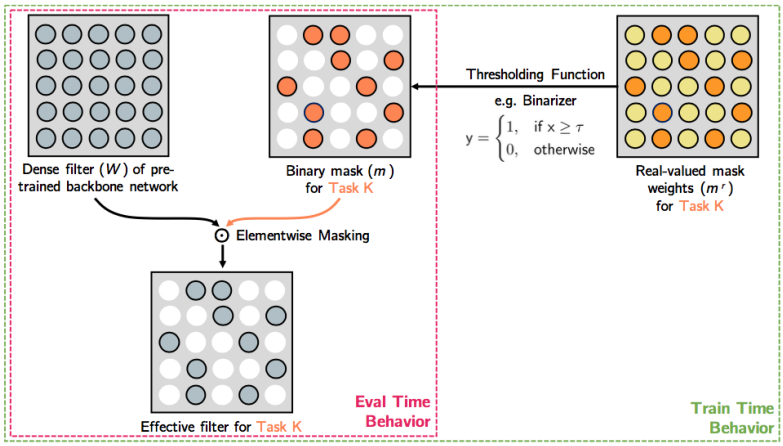


Figure 4.13. *Overview of Piggyback method fo learning piggyback masks for fixed backbone networks. During training, maintaining a set of real-valued weights* m^r *which are passed through a thresholding function to obtain binary-valued masks* m*. These masks are applied to the weights* W *of the backbone network in an element-wise fashion, keeping individual weights active, or masked out. The gradients obtained through backpropagation of the task-specific loss are used to update the real-valued mask weights. After training, the real-valued mask weights are discarded, and only the thresholded mask is retained, giving one network mask per task. Source: [18]*

LRP operates by propagating the prediction f(x) backward in the neural network through purposely designed local propagation rules. The propagation procedure implemented by LRP is subject to a conservation property, where what has been received by a neuron must be redistributed to the lower layer in equal amount as it is shown in Figure 4.14.

LRP can be implemented in many different ways:

- Uniform LRP-0: picks many local artifacts of the function. The explanation is overly complex and does not focus sufficiently on the actual concepts. The explanation is neither faithful nor understandable.
- Uniform LRP-ε: removes noise elements in the explanation to keep only a limited number

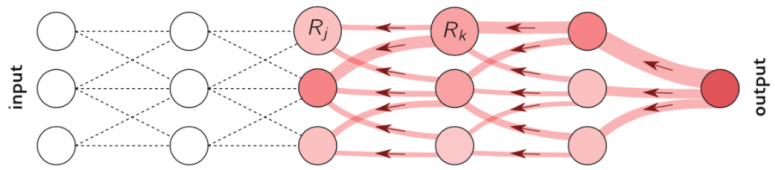


Figure 4.14. *Illustration of the LRP procedure. Each neuron redistributes to the lower layer as much as it has received from the higher layer. Source: [19]*

of features that match the actual concepts in the image. It is a faithful explanation but too sparse to be easily understandable.

- Uniform LRP-γ: is easier for a human to understand because features are more densely highlighted, but it also picks unrelated concepts and makes it unfaithful.
- Composite LRP: overcomes the disadvantages of the approaches above.

Montavon et al. [19] implement the following algorithms in the VGG-16 [85] achieving the results presented on Figure 4.15.

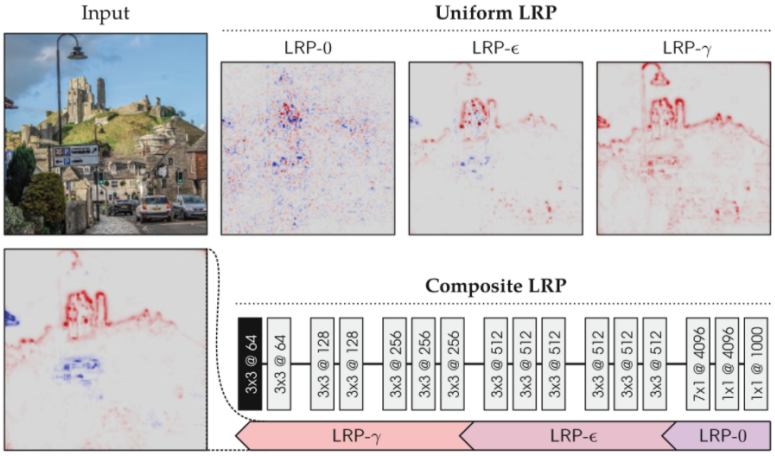


Figure 4.15. *Input image and pixel-wise explanations of the output neuron 'castle' obtained with various LRP procedures. Parameters are $\epsilon = 0.25$ std and $\gamma = 0.25$. Source: [19]*

Another well-known XAI technique is LIME, Local Interpretable Model-agnostic Explanations [86], a model-agnostic explanation approach that returns explanations as features importance vectors. The main idea of LIME is that the explanation may be derived locally from records generated randomly in the neighborhood of the instance that has to be explained.

Chapter 5

Back to the Future: A transfer learning approach for structured pruning

5.1 Abstract

Large Deep Learning models are overparameterized, and thus they can be pruned in both magnitude and structured aspects. In the field of Natural Language Processing, Transformer-based architectures were examined under self-attention head pruning techniques and achieved promising results. In this study, we examine a structured pruning approach for BERT-based architectures that implies that in transfer learning, the final model is the result of a fine-tuning process of a pre-trained model. Thus, we consider both the pre-trained and the fine-tuned model to prune attention heads. Furthermore, we study this method through the Lottery Ticket Hypothesis, where we see that considering both the pre-trained and the fine-tuned model outperforms the approach which only considers the fine-tuned model. Moreover, we propose a better application of the Lottery Ticket Hypothesis in structured pruning named "Iterative Structured Pruning". Finally, we examine our technique on another modality, more precisely in Automatic Speech Recognition through wav2vec 2.0, and we see corresponding results.

5.2 Introduction

Recently the field of Natural Language Processing (NLP) has faced significant progress in many NLP tasks, such as Natural Language Understanding (NLU) and Natural Language Inference (NLI). A significant part of this progress is due to the Transfer Learning approach of the Transformer-based models proposed by Vaswani et al. [7].

One of the most famous Transformer-based models is BERT, proposed by Devlin et al. [10]. BERT is a pre-trained language representation model trained from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

Transformer-based models contain millions of parameters that slow down the interference, increase the memory footprint, the number of computation operations (FLOPs), the power usage and cause problem-related environmental issues. This problem tends to rapidly scale up: BERT-base [21] contains 110 million parameters, Turing-NLG [22] contains 17 billion parameters and the GPT-3 [23] includes 175 billion.

A response to this problem is model compression. Many researchers apply pruning techniques on BERT-base models by pruning weights or structured components such as attention heads. Many researchers, like Voita et al. [14], Kovaleva et al. [13] and Michel et al. [1], suggest that

Transformer-based models are heavily overparameterized by removing a large number of heads without a significant performance trade-off.

Although these works achieve significant pruning rates, they do not consider that the model is produced through a fine-tuning process and thus is highly dependent on the pre-trained model. This study examines a new heuristic for indicating the attention head's importance of the model based on both the pre-trained and the fine-tuned model. We test our method on several GLUE benchmarks [11], on the SciERC dataset [24] and on the PubMed 200k RCT dataset [25].

We investigate the effect of the pre-trained domain in our pruning technique, where we replace our pre-trained model, and we use SciBERT [33]. SciBERT is pre-trained on large scientific-related corpora, and we see that our approach works regardless of the pre-trained domain.

Furthermore, we study this method through the Lottery Ticket Hypothesis (LTH), where we see that considering both the pre-trained and the fine-tuned model for obtaining initialization mask for LTH outperforms the approach which only considers the fine-tuned model. We also propose a better pipeline for structured pruning with an "Iterative Structured Pruning" methodology.

Finally, we try structured pruning on a different modality and more precisely in the area of Automatic Speech Recognition through wav2vec 2.0 [26]. Through this experiment, we see that our methodology works well in other modalities.

5.3 Related Work

Many studies have shown that BERT is overparameterized [27] and try different compression techniques, including magnitude pruning by and Gordon et al. [28] and structured pruning by Voita et al. [14], Kovaleva et al. [13] and Michel et al. [1].

Voita et al. [14] suggest that all but a few Transformer heads could be pruned without significant losses in performance. For BERT, Clark et al. [27] observe that most heads in the same layer have similar self-attention patterns (perhaps related to the fact that the output of all self-attention heads in a layer is passed through the same MLP), which explains why Michel et al. [1] were able to reduce most layers to a single head.

Pruning strategies that consider both the fine-tuned and the pre-trained model were not proposed in the Natural Language Processing bibliography until Sanh et al. [29] propose a pruning approach named "movement pruning" that prunes BERT weights based on the change of the weight value during fine-tuning.

Frank et al. [12] propose the Lottery Ticket Hypothesis, which is formulated as follows: "dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that - when trained in isolation - reach test accuracy comparable to the original network in a similar number of iterations". Frank et al. not only propose a pruning pipeline but also propose the reason why deep neural networks work.

Chen et al. [16] apply the magnitude approach of the Lottery Ticket Hypothesis on BERT and achieve excellent results. Their study suggest that the Lottery Ticket Hypothesis can be applied to a not randomly initialized network because BERT is a pre-trained network. Prasanna et al. [15] apply a structured approach where the importance of a head is calculated by the sensitivity score proposed by Michel et al. and achieve similar results. Both Chen et al. and Prasanna et al. find that such winning subnetworks exist and transferability between subnetworks for different tasks varies.

5.4 Problem Definition

5.4.1 Structured Pruning Definition

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$ and a desired sparsity level κ, neural network structured pruning can be written as the following constrained optimization problem:

$$\min_{\mathbf{w_s}} L(\mathbf{w_s}; \mathcal{D}) = \min_{\mathbf{w_s}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(\mathbf{w_s}; (\mathbf{x}_i, \mathbf{y}_i)\right)$$
$$\text{s.t.} \quad \mathbf{w_s} \in \mathbb{R}^m, \quad \|\mathbf{w_s}\|_0 \leq \kappa$$

Here, $\ell(\cdot)$ is the standard loss function, $\mathbf{w_s}$ is the structured set of parameters of the neural network e.g. attention heads, m is the total number of structured sets and $\|\cdot\|_0$ is the standard L_0 norm.

There is no efficient way to minimize the L_0 norm as it is non-convex, NP-hard, and requires combinatorial search. Thus, Structured Pruning is an NP-hard problem.

Structured pruning can take place before, during, and afterward fine-tuning. In this work, we first fine-tune the model, and afterward, we apply our pruning methodology.

5.4.2 BERT Architecture

BERT is fundamentally a stack of Transformer encoder layers Vaswani et al. [7]. All layers have an identical structure: a multi-head self-attention (MHAtt) block followed by an MLP, with residual connections around each.

MHAtt consists of N_h independently parameterized heads. An attention head h in layer l is parameterized by $W_k^h, W_q^h, W_v^h \in \mathbb{R}^{d_h \times d}, W_o^h \in \mathbb{R}^{d \times d_h}$ and d_h is typically set to d/N_h. Given n d-dimensional input vectors $\mathbf{x} = x_1, x_2, \ldots x_n \in \mathbb{R}^d$, MHAtt is the sum of the output of each individual head applied to input x :

$$\text{MHAtt}(\boldsymbol{x}, q) = \sum_{h=1}^{N_h} \text{Att}_{W_k^h, W_q^h, W_v^h, W_o^h}(\boldsymbol{x}, q)$$

To allow the different attention heads to interact with each other, Transformers apply a non-linear feed-forward network over the MHAtt output at each Transformer's layer [30]. Each attention head may pay attention to different functionalities, such as syntax and semantics.

5.5 Proposed Method

While pruning is highly effective for standard supervised learning, it is less helpful in the transfer learning approach. In supervised learning, weight values are primarily determined by the end task. In transfer learning, weight values are determined mainly by the pre-trained model's weights which are only fine-tuned on the end task. In this work, we argue that to effectively reduce the size of models for transfer learning, one should use pruning strategies that consider both the pre-trained and the fine-tuned model.

Firstly, as Michel et al. [1] proposed, we introduce mask variables ξ_h with values in $\{0, 1\}$, where $\xi_h = 1$ denotes that the corresponding head h is not masked while $\xi_h = 0$ denotes that head h is masked. This leads to a modification of the formula for MHAtt:

$$\text{MHAtt}(\boldsymbol{x}, q) = \sum_{h=1}^{N_h} \xi_h \text{Att}_{W_k^h, W_q^h, W_v^h, W_o^h}(\boldsymbol{x}, q)$$

We define the expected absolute sensitivity of the model to the mask variables ξ_h:

$$I_h = \mathbb{E}_{x \sim X} \left| \frac{\partial(\alpha * L_1(x) + (1 - \alpha) * L_2(x))}{\partial \xi_h} \right|$$

where X is the data distribution, $L_1(x)$ the loss of the fine-tuned model on sample x, $L_2(x)$ the loss of the original model on sample x and $\alpha \in [0, 1]$ is an "specialization factor", which controls the importance of each loss. More precisely, $\alpha > 0.5$ denotes that the pruning process will pay more attention in the fine-tuned model, while $\alpha < 0.5$ denotes the opposite and $\alpha = 0.5$ denotes that the pruning algorithm will equally focus to both models. The implementation of the pruning methodology with importance score can be described by the Algorithm 5.1.

Algorithm 5.1: *Structured Pruning with Importance Score*

1: Fine-tune the pre-trained network
2: Set the initial pruning mask of attention heads to $s = 1^d$ ▷ d: dimension
3: **repeat**
4: Calculate I_h for the non-pruned attention heads
5: Sort heads in descending order based on I_h
6: Prune κ% of initial heads with the lowest I_h and update s
7: **until** the sparsity of s reaches s_T ▷ s_T: Sparsity Threshold
8: **return** s

After extracting the mask s, we can perform a one-shot structured Lottery Ticket Hypothesis on BERT: use the mask s on the pre-trained BERT and fine-tune it to the given task. Alternatively, we can use an Iterative Structured Pruning approach for Lottery Ticket Hypothesis. This approach is described in the Algorithm 5.2.

Algorithm 5.2: *Iterative Structure Pruning (ISP)*

1: Fine-tune the pre-trained network
2: Set the initial pruning mask of attention heads to $s = 1^d$ ▷ d: dimension
3: **repeat**
4: Calculate I_h for the non-pruned attention heads
5: Sort heads in descending order based on I_h
6: From the pre-trained model prune κ% of remaining heads with the lowest I_h and update s
7: Fine-Tune the pre-trained network
8: **until** the sparsity of s reaches s_T ▷ s_T: Sparsity Threshold
9: **return** s

Through the Iterative Structure Pruning approach, the final model is fine-tuned only one time.

5.6 Experiments

5.6.1 Different Pruning Rates

All experiments in this section are done on the pre-trained BERT model ("bert-base-uncased", 12-layers, 768-hidden, 12-heads, 110M parameters) from the Transformers library [31]. The pre-trained model is fine-tuned with metrics shown in Table 5.1 on 6 different GLUE tasks:

- MNLI: Multi-Genre Natural Language Inference Corpus [74]
- QQP: the Quora Question Pairs dataset
- QNLI: Question-answering NLI based on the Stanford Question Answering Dataset [75]

Dataset	MNLI	QQP	QNLI	MRPC	SST-2	CoLA
Train Ex.	392,704	363,872	104,768	3,680	67,360	8,576
Iters/Epoch	12,272	11,371	3,274	115	2,105	268
Epochs	3	3	3	3	3	3
Batch Size	32	32	32	32	32	32
Learning Rate	2×10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-5}	2×10^{-5}
Optimizer	AdamW with $\epsilon = 1 \times 10^{-8}$					
Eval Metric	Matched Acc.	Accuracy	Accuracy	Accuracy	Accuracy	Matthew's

Table 5.1. *GLUE tasks [20], dataset sizes metrics and fine-tuning hyperparameters reported in this study*

α	MNLI	QNLI	QQP	SST-2	MRPC	CoLA
1	0.586	0.564	0.725	0.837	0.484	0.183
0.7	0.589	**0.670**	0.719	0.783	0.589	0.202
0.6	0.586	0.603	**0.750**	0.798	**0.652**	0.206
0.5	0.566	0.594	0.694	**0.843**	0.537	**0.287**
0.4	**0.623**	0.599	0.709	0.832	0.571	0.229

Table 5.2. *Results on GLUE tasks after pruning* 112 *heads. We conduct the experiments for different values of the hyperparameter α, and the result is the model's performance in the development set after pruning* 112 *attention heads. For $\alpha = 1$ our method is equivalent to the baseline method of Michel et al. [1]. For each task, the best result is bolded. For each experiment, we test 4 random seeds.*

- MRPC: Microsoft Research Paraphrase Corpus [72]
- SST-2: Stanford Sentiment Treebank [71]
- CoLA: Corpus of Linguistic Acceptability [70]

For the fine-tuning process, all the training data of each benchmark are used. After fine-tuning, the proposed algorithm 5.1 runs for different pruning rates with $\kappa = 10\%$ on the development set of each benchmark. Finally, the pruned model is tested on the development set, as the test sets of GLUE benchmarks are not publicly distributed. Testing on the development set of GLUE is a common practice used in many BERT pruning papers [15], [16]. In the following experiments, we see the impact of different methodologies on different pruning rates. The following experiments are the average result of 4 different runs with different random seeds.

Deep Pruning Rate. As shown in previous studies [15], [14] BERT-base models can be heavily pruned, so in this section, we perform a deep pruning investigation. We perform 8 iterations of the proposed algorithm, and in each iteration, we prune 14 heads, so in total, we prune 112 out of 144 heads, which is approximately the 80% of BERT attention heads. The results can be described from Table 5.2.

From these experiments, we can suggest that for deep pruning rates, our heuristic outperforms the baseline heuristic proposed by Michel et al. [1] for every examined GLUE task. We can also see that for the majority of the hyperparameter α values, the final model performs better than the baseline model. This observation indicates that in deep pruning rates, the pre-trained model scores with higher importance heads scored with lower importance on the fine-tuned model. Thus, the final importance score, which is calculated as a weighted average between these two scores, gives these heads a higher importance compared to the baseline, and that is beneficial as it is shown in Table 5.2. Thus, we can conclude that considering both the pre-trained and the fine-tuned model during the pruning stage is beneficial. The best value of the hyperparameter α depends on the nature of the task and will be further investigated in the following experiments.

MNLI	QNLI	QQP	SST-2	MRPC	CoLA
0.717($\alpha = 1$)	0.734	0.791	0.875	0.639	0.286
0.732 ($\alpha = 0.4$)	**0.787** (0.7)	**0.811** (0.6)	**0.878** (0.4)	**0.730** (0.6)	**0.387** (0.5)

Table 5.3. *Results on GLUE tasks after pruning 98 heads. The first line describes the performance of the baseline for $\alpha = 1$, and then we present the best performance of the model in the development set after pruning 98 attention heads. For each task, the best result is bolded, and the corresponding best α is mentioned. For each experiment, we test 4 random seeds.*

Lighter Pruning Rate. In this section, we perform a lighter pruning investigation; thus, we perform 7 iterations of the proposed algorithm, and in each iteration, we prune 14 heads, so, in total, we prune 98 out of 144 heads, which is approximately 70% of BERT attention heads. The results can be described from Table 5.3.

From the above experiments, we see that our proposed methodology outperforms the baseline even for a lighter pruning rate. We also see that, for each benchmark, the best performance is achieved with the same α as the deep pruning rate experiment. So, we can suggest that our approach is robust regarding α in these pruning rates. Overall, we see that considering both the pre-trained and the fine-tuned model is beneficial for the pruning process.

Different Pruning Rates. In this section, we perform pruning investigation for different pruning rates, and we present the results of the final model with a pruning step equal to 14 heads. In total, we perform 8 pruning steps, and the result is the average of 4 different runs with different random seeds. The results can be described from Figure 5.1.

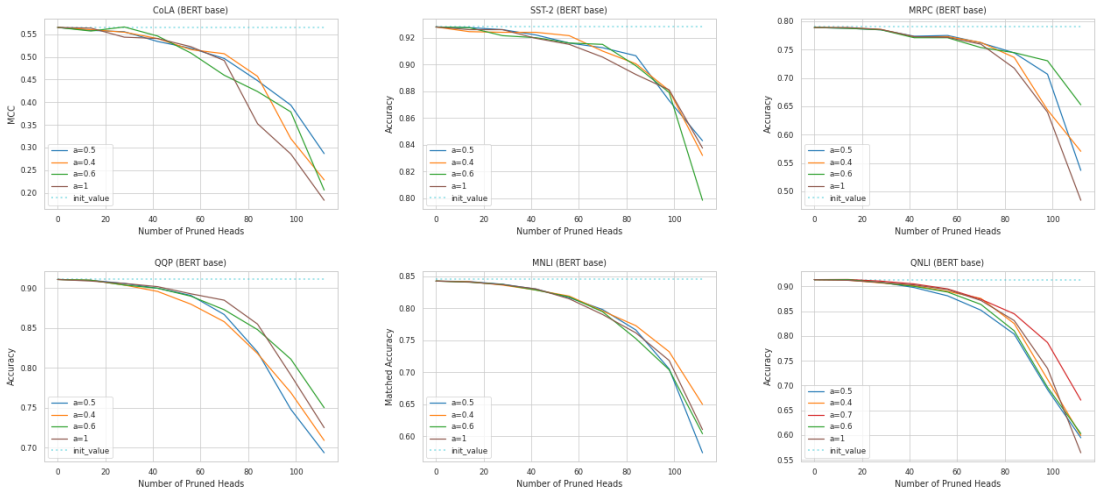


Figure 5.1. *Results on GLUE tasks. We conduct the experiments for different values of the hyperparameter α, and the result is the performance of the model in the development set after pruning 14 attention heads in each pruning step. For $\alpha = 1$ our method is equivalent to the baseline method of Michel et al. [1]. For each experiment, we test 4 random seeds.*

From Figure 5.1 we see that for lower pruning rates, all the methodologies regardless of the value of α achieve similar performance, which indicates that in the first 5 pruning iterations, the effect of each head removal is similar. This observation indicates that in these pruning rates, some heads can carry over the work of pruned heads. This result agrees with the results of other pruning studies on BERT, such as Clark et al. [27], Prasanna et al. [15].

Moreover, we can see that for greater pruning rates, our approach generally outperforms the baseline for any given value of α. A result discussed extensively in the experiments above.

5.6.2 The specialization factor α

As we see from the previous chapter, in the first pruning iterations, α does not make any significant difference in the result, but finding the best value for α in deeper pruning rates is crucial. In this section, we are going to further investigate the result of α in the pruning process by answering the following questions: "Does the best value of α depend on the end task?", "Does the best value of α depend on the pre-trained corpora?".

The proposed importance score includes two importance scores: the importance score of the pre-trained model and the importance score of the fine-tuned model. The final score is a weighted average between these two scores, and the weight of each score is controlled through α. We assume that α during pruning controls the amount of weight contributed to the fine-tuned model. Thus, $\alpha < 0.5$ suggests that the pruning process should give more weight to the pre-trained model, and $\alpha > 0.5$ suggests the pruning procedure should be more specialized on the fine-tuned model.

Does the best value of α depend on the end task? - Models Fine-tuned on Scientific Domain. In order to examine the effect of specialization factor,a, we examine BERT-base models fine-tuned on datasets of Computer and Biomedical Science. BERT-base has a small vocabulary overlap between scientific domain as examined by Gururangan et al. [32] in the Figure 5.2, and we want to see the effect of α in datasets with domains different from the pre-trained domains.

Figure 5.2. *Vocabulary overlap (%) between domains. PT denotes a sample from sources similar to BERT pretraining corpus. Vocabularies for each domain are created by considering the top 10K most frequent words (excluding stopwords) in documents sampled from each domain.*

All experiments in this section are done on the "bert-base-uncased" model from the Transformers library [31]. It is fine-tuned on two different Scientific Domain Datasets: SciERC [24] and PubMed 20k RCT [25] with metrics shown in Table 5.5. Then the proposed algorithm 5.1 is implemented with $\kappa = 10\%$.

Firstly, we perform 8 iterations of the proposed algorithm, and in each iteration, we prune 14 heads, so in total, we prune 112 out of 144 heads, which is approximately 80% of BERT attention heads. The results are shown in Table 5.4.

We also perform pruning investigation for different pruning rates, and we present the results of the final model with a pruning step equal to 14 heads. In total, we perform 8 pruning steps. The results can be described from Figure 5.3.

From Figure 5.3 we see that for lower pruning rates, all the methodologies regardless of the value of α achieve similar performance, which indicates that in the first pruning iterations, the impact of each head removal is similar. Nevertheless, for the following iterations, we see that our approach for $\alpha = 0.6$ outperforms the baseline, as it can be clearly shown from the Table 5.4.

From Figure 5.3 we can conclude that the best score for both Scientific Domain tasks is $\alpha = 0.6$.

α	PubMed 20k RCT	SciERC
1	0.454	0.511
0.4	0.329	0.473
0.5	0.605	0.547
0.6	**0.736**	**0.557**

Table 5.4. *Results on PubMed 20k RCT and SciERC after pruning* 112 *heads. We conduct the experiments for different values of the hyperparameter* α*, and the result is the model's performance after pruning* 112 *attention heads. For* $\alpha = 1$ *our method is equivalent to the baseline method of Michel et al. [1].*

Dataset	PubMed 20k RCT	SciERC
Train Ex.	178,144	3,216
Iters/Epoch	11,134	201
Epochs	3	3
Batch Size	16	16
Learning Rate	2×10^{-5}	2×10^{-5}
Optimizer	AdamW with $\epsilon = 1 \times 10^{-8}$	
Eval Metric	Accuracy	Accuracy

Table 5.5. *PubMed 20k RCT and SciERC dataset sizes metrics and fine-tuning hyperparameters reported in this study*

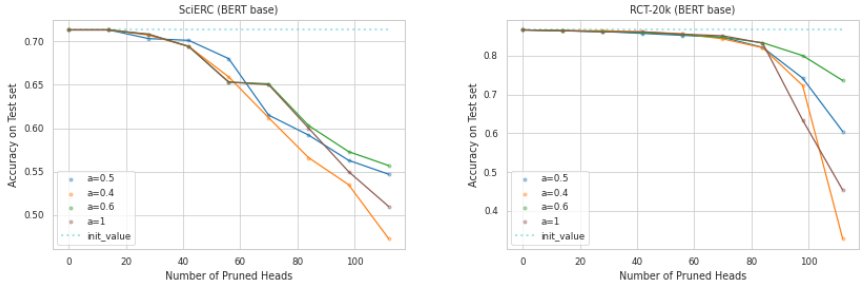


Figure 5.3. *Results on PubMed 20k RCT and SciERC on different pruning rates. We conduct the experiments for different values of the hyperparameter* α*, and the result is the model's performance after pruning* 14 *attention heads in each pruning step. For* $\alpha = 1$ *our method is equivalent to the baseline method of Michel et al. [1].*

This observation can be justified because BERT pre-trained corpora have small vocabulary overlap with scientific domain datasets. Thus, pruning should focus more on the fine-tuned model, and the pruning procedure should be more specialized on the fine-tuning model.

Does the best value of α depend on the pre-trained corpora? - Models with Scientific Pre-trained Domain.

As Gururangan et al. [32] suggest, BERT-base has small vocabulary overlap regarding scientific domains. So, for this section, we use another pre-trained model, trained on numerous scientific-related corpora. This model is called SciBERT [33] and we use the implementation from the Transformers library [31].

We fine-tuned our model on 5 GLUE tasks: MNLI, QNLI, MRPC, SST-2, and CoLA and a Scientific Domain Dataset: PubMed 20k RCT. We perform 8 iterations of the proposed algorithm, and in each iteration, we prune 14 heads, so we prune 112 out of 144 heads, which approximately prune 80% of BERT attention heads. We perform the experiments for many values of the hyperparameter α, including for $\alpha = 1$, which is equivalent to the Michel et al. [1] method used as the baseline. The best value for α can be found in Table 5.6.

A more in detail description of this experiment is presented in Figure 5.4, where we prune 14

	MNLI	QNLI	SST-2	MRPC	CoLA	PubMed 20k RCT
winning-α	0.7	0.6	0.6	0.7	1	0.6

Table 5.6. *Results on GLUE tasks and PubMed 20k RCT after pruning* 112 *heads. We prune* 112 *heads from the models with different values of the hyperparameter* α*. Then we test their performance, and for the best performance, we mention the corresponding* α *in this Table.*

heads in each iteration, and in total, we prune 112 heads.

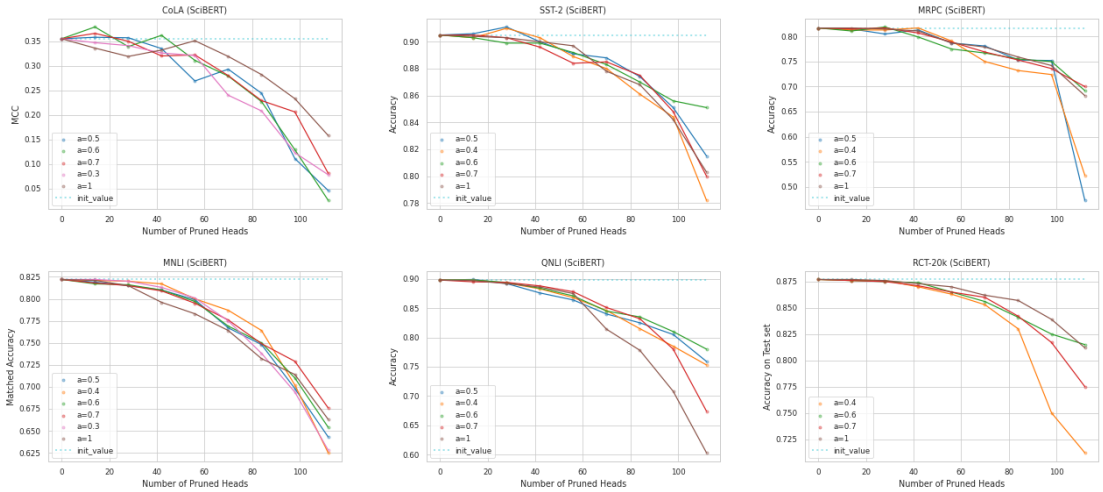


Figure 5.4. *Results on GLUE tasks and PubMed 20k RCT on different pruning rates. We conduct the experiments for different hyperparameter* α *values, and the result is the model's performance after pruning* 14 *attention heads in each pruning step.*

GLUE benchmarks have a small vocabulary overlap with the Scientific related corpora on which SciBERT is trained. Therefore, we suggest that the heads will be drastically changed and move closer to the scientific domain distribution through fine-tuning. Thus, we expect that the best pruning strategy will be performed with a value of α greater than 0.5, indicating that the pre-trained model contains some valuable general knowledge, but the importance score should be mainly calculated from the fine-tuned model. Indeed, from Figure 5.4 we see that the best pruning strategy is performed with the specialization factor set equal to $\alpha = 0.6$.

From Figure 5.4 it is worth mentioning the plot of CoLA dataset, which is the only dataset in which the winning methodology uses only the fine-tuned model to calculate the importance scores. Moreover, the absolute performance of each GLUE task is worse when SciBERT is used as a pre-trained model.

PubMed 20k RCT is a scientific-related dataset with significant vocabulary overlap with the Scientific related corpora on which SciBERT is trained. We see the best value for $\alpha = 0.6$, and the absolute performance after pruning is better when the model uses a pre-trained model trained on scientific text.

5.6.3 Do we win the Lottery?

In the previous sections, we first fine-tune the model and then apply a pruning methodology which creates a set of head masks and a final pruned model. However, this process does not take advantage of the re-training abilities of the BERT model. Indeed, we can use the set of head masks as initialization masks on the pre-trained model and then fine-tune it once.

This idea is a structured approach of Lottery Ticket Hypothesis proposed by Frankle et al. [12]. In previous studies, they apply this idea on BERT [15], but in our work, we see that our

a	MNLI	QNLI	MRPC	SST-2	CoLA
1	0.817	0.820	0.773	0.903	0.329
0.6	**0.824**	0.874	0.778	0.913	0.334
0.5	0.819	0.876	0.607	0.917	**0.435**
0.4	0.809	**0.880**	**0.783**	**0.918**	0.312

Table 5.7. *Results on GLUE tasks*

approach, which creates the set of head masks regarding both the pre-trained and the fine-tuned model, outperforms the approach which only considers the fine-tuned model.

Overall, we follow the following pipeline: we fine-tune the pre-trained model, and then we apply our proposed pruning algorithm with $\kappa = 10\%$ for 8 iterations to extract a set of 112 head masks. Then we apply this set of head masks on the pre-trained BERT model ("bert-base-uncased", 12-layers, 768-hidden, 12-heads, 110M parameters) from the Transformers library [31]. In this way, we have a pre-trained model where 112 heads are masked.

Then we fine-tune each model to the corresponding GLUE task with the metrics shown in Table 5.1. We perform this experiment for 5 GLUE Tasks: MNLI, QNLI, MRPC, SST-2, and CoLA.

The results can be found in Table 5.7. Our approach outperforms the models with a set mask produced by Michael et al. [1] heuristic, while we prune approximately 80% of the attention heads and we have a small drop on the initial model performance. This observation indicates that considering both the pre-trained and the fine-tuned model is beneficial for a Lottery Ticket Hypothesis approach.

5.6.4 Iterative Structured Pruning

This section conducts the Iterative Structured Pruning (ISP) approach as described in the Methodology Section. This approach contributes to the field of Lottery Ticket Hypothesis because it is the first time that structured pruning is conducted iteratively, and it generalizes Chen et al. [16] iterative approach.

We conduct the iterative pruning technique in these experiments by first fine-tuning the pre-trained BERT model on the downstream task. Then we detect the 14 heads with the lowest score as measured from our heuristic with $\alpha = 1$. Then we take the pre-trained model, prune those 14 heads, and fine-tune the model. Thus, the final model is fine-tuned once with the 90% of attention heads remaining. We can continue this process by detecting the 14 heads with the lowest score in this pruned model. Then we take the pre-trained model, prune those $14 + 14$ heads, and then fine-tune the model. Thus, the final model is fine-tuned once with the 80% of attention heads remaining. In this method, the final pruned model is fine-tuned once.

We compare our results with the not Iterative Pruning, which is conducted by pruning $x\%$ of heads in the pre-trained model and then fine-tuning the model. The final model has $100 - x\%$ of attention heads remaining.

With the Iterative Structured Pruning approach, the model's overall performance is more outstanding than the not Iterative Pruning approach. This observation can be described by the Figure 5.5 where we conduct these experiments on 2 GLUE Tasks: MNLI and QNLI. This approach indicates a trade-off between performance and computational cost.

5.6.5 Pruning in a different modality

In the previous sections, we prune models specialized in the language modality. This section investigates whether our approach can be implemented for pruning BERT-base models trained on other modalities.

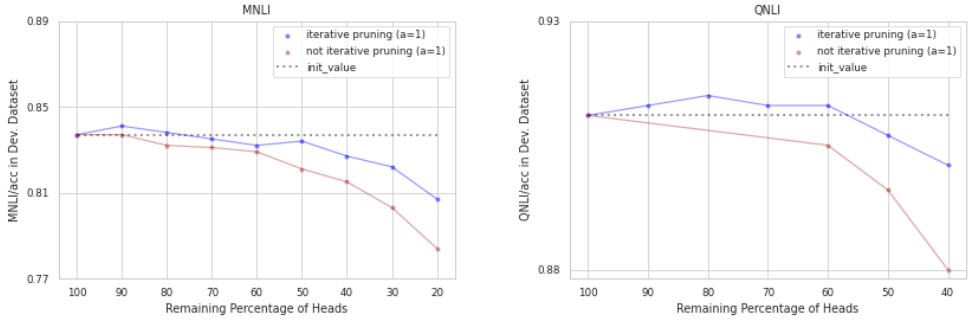


Figure 5.5. *Iterative Structured Pruning (ISP) on MNLI and QNLI with* $\alpha = 1$

More specifically, we examine speech modality through the task of Automatic Speech Recognition. For this purpose, we use the newly introduced BERT-base model wav2vec 2.0 [26]. Wav2vec 2.0 masks the speech input in the latent space and solves a contrastive task defined over a quantization of the latent representations, which are jointly learned and thought it learns powerful representations.

We fine-tune the pre-trained models for phoneme recognition on the TIMIT dataset [2]. It contains five hours of audio recordings with detailed phoneme labels. We use the standard train, development, and test split and follow the standard protocol of collapsing phone labels to 39 classes. Thus performance is computed in Word Error Rate (WER).

Then we apply our proposed pruning methodology and we perform pruning investigation for different pruning rates. In total, we perform 8 pruning steps, and in total, we prune 112. The results can be described from Figure 5.6.

From Figure 5.6 we see that for lower pruning rates, all the methodologies regardless of the value of α achieve similar performance, which indicates that in the first 4 pruning iterations, some heads can carry over the work of pruned heads.

Moreover, we can see that our approach outperforms the baseline for any given value of α for the greatest pruning rates.

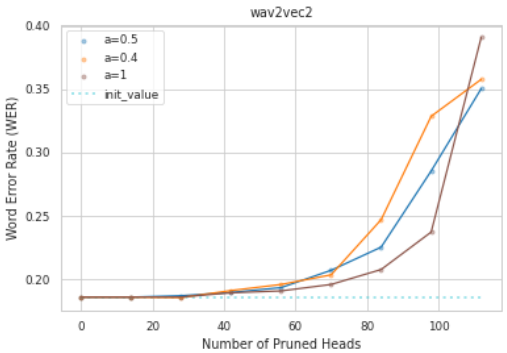


Figure 5.6. *wav2vec 2.0 fine-tuned on Timit dataset. The Performance is computed in WER.*

Chapter 6

Conclusions

In this Diploma Thesis, we study the compression of Deep Neural Networks, and more precisely, we study the structured pruning in Natural Language Processing models. From our work, we draw some conclusions that can be divided into two main aspects:

- **Studying pruning for pruning.** We propose a better implementation of pruning that considers both the pre-trained and the fine-tuned model.
- **Studying pruning for a better understanding of the model.** We see that through fine-tuning, the model forgets prior knowledge, and in order to overcome this problem, we study both pre-trained and fine-tuned models for better pruning results.

6.1 Discussion

In Section 5.5 Proposed Method, we extend the work from Prasanna et al. [15], and Michel et al. [1] and we propose a new heuristic for calculating the importance score of self-attention heads in BERT-based models. These models are created after a fine-tuning process, where weight values are mostly predetermined by the original model and are only fine-tuned on the end task. Thus, pruning strategies should consider the pre-trained and the final fine-tuned model, and the head importance score should be calculated considering both the importance of the pre-trained and the fine-tuned head. At the same time, the amount of specialization, which corresponds to "how much importance should we pay to the final fine-tuned model," is controlled by a tunable hyperparameter a.

This study performs pruning investigations for different pruning rates on BERT-base models fine-tuned on different GLUE datasets. We conclude that our approach, in general, outperforms the baseline for any given value of a for greater pruning rates.

In order to examine the effect of specialization factor a, we examine BERT-base models fine-tuned on datasets of Computer and Biomedical Science. We suggest that a greater value for a gives better performance on the final model. This observation can be justified because BERT pre-trained corpora have small vocabulary overlap with the scientific domain datasets. Thus, pruning should focus more on the fine-tuned model, and the pruning procedure should be more specialized on the fine-tuning model.

Furthermore, in order to understand whether the best value of α depends on the pre-trained corpora we use as pre-trained models, networks with Scientific Pre-trained Domains. Indeed, we see a correlation between the pre-trained corpora distribution and the value of specialization factor a.

Moreover, we investigate a structured approach to the Lottery Ticket Hypothesis. We show that our approach that creates the set of head masks regarding both the pre-trained and the fine-tuned model outperforms the approach that only considers the fine-tuned model.

Regarding the Lottery Ticket Hypothesis, we propose a better implementation for structured pruning through an Iterative Structured Pruning (ISP) approach. This approach contributes to the Lottery Ticket Hypothesis because it is the first time that structured pruning has been conducted iteratively.

Finally, we investigate whether our approach can be implemented for pruning BERT-base models trained on other modalities. More specifically, we examine speech modality through the task of Automatic Speech Recognition. For this purpose, we use the newly introduced BERT-base model, wav2vec 2.0, and we show corresponding results. This study is critical because, as we can conclude from the literature, many Transformer-based architectures achieve significant results on different modalities such as speech and vision.

After all, investigating pruning is essential for reducing the model's size and achieving interaction speed-up. Structured pruning examines the importance of the head and is a way to conclude a better understanding of the model. By classifying heads based on their importance, we identify which heads work better in the end task, thus understanding why BERT-based models work. Indeed, as Voita et al. [14] said, "Specialized Heads Do the Heavy Lifting" and by understanding which heads are "specialized" and what makes some heads specialize, we can create better implementations for BERT-based models.

6.2 Future Work

Through the end of this thesis, we wish to open some new roads and create new aspects to explore pruning mechanisms and the Lottery ticket Hypothesis. We suggest the following points be explored in future work:

- **Better understanding of the fine-tuning process.** The proposed fine-tuning process of Transformer-based models [7] seems to create models that forget the pre-trained knowledge. The pre-trained knowledge is generic and essential for many Natural language Processing Tasks, and thus our pruning approach that considers both models yields performance. We suggest that this work opens the road for better fine-tuning methodologies.

- **Structured Pruning of other structures in Transformers.** Based on Prasanna et al. [15] study, we know that other structures inside Transformers-based models can be pruned, for example, fully connected layers. Perhaps our approach could be implemented for a better pruning algorithm in these fully connected layers.

- **Regarding Iterative Structured Pruning.** The iterative structured pruning algorithm could work in other Transformer-based models specialized in other modalities and with other deep learning architectures such as Convolutional Neural Network.

- **Pruning Models of Different Modalities.** To the best of our knowledge, this is the first work that examines pruning on models specialized in modalities different than language. As a result, pruning Deep Leaning models of different modalities will achieve better interference time and a better understanding of these models.

Bibliography

[1] Paul Michel, Omer Levy και Graham Neubig. *Are Sixteen Heads Really Better than One? arXiv:1905.10650 [cs]*, 2019. arXiv: 1905.10650.

[2] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett και N. L. Dahlgren. *DARPA TIMIT Acoustic Phonetic Continuous Speech Corpus CDROM*, 1993.

[3] Ian Goodfellow, Yoshua Bengio και Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] Richard O. Duda, Peter E. Hart και David G. Stork. *Pattern Classification*. Wiley, New York, 2η έκδοση, 2001.

[5] Sneha Chaudhari, Varun Mithal, Gungor Polatkan και Rohan Ramanath. *An Attentive Survey of Attention Models*, 2021.

[6] Tianyang Lin, Yuxin Wang, Xiangyang Liu και Xipeng Qiu. *A Survey of Transformers*, 2021.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser και Illia Polosukhin. *Attention Is All You Need. arXiv:1706.03762 [cs]*, 2017. arXiv: 1706.03762.

[8] Tomas Mikolov, Kai Chen, Greg Corrado και Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*, 2013.

[9] P. Vincent Y. Bengio, R. Ducharme. *A neural probabilistic language model. Journal of Machine Learning Research*. 2003.

[10] Jacob Devlin, Ming Wei Chang, Kenton Lee και Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs]*, 2019. arXiv: 1810.04805.

[11] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy και Samuel R. Bowman. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*, 2019.

[12] Jonathan Frankle και Michael Carbin. *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*, 2018.

[13] Olga Kovaleva, Alexey Romanov, Anna Rogers και Anna Rumshisky. *Revealing the Dark Secrets of BERT. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, σελίδες 4364–4373, Hong Kong, China, 2019. Association for Computational Linguistics.

[14] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich και Ivan Titov. *Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. arXiv:1905.09418 [cs]*, 2019. arXiv: 1905.09418.

[15] Sai Prasanna, Anna Rogers και Anna Rumshisky. *When BERT Plays the Lottery, All Tickets Are Winning. arXiv:2005.00561 [cs]*, 2020. arXiv: 2005.00561.

[16] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang και Michael Carbin. *The Lottery Ticket Hypothesis for Pre-trained BERT Networks. arXiv:2007.12223 [cs, stat]*, 2020. arXiv: 2007.12223.

[17] Arun Mallya και Svetlana Lazebnik. *PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning*, 2018.

[18] Arun Mallya, Dillon Davis και Svetlana Lazebnik. *Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights*, 2018.

[19] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek και Klaus Robert Müller. *Layer-Wise Relevance Propagation: An Overview*, σελίδες 193–209. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag, 2019.

[20] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy και Samuel R. Bowman. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. arXiv:1804.07461 [cs]*, 2019. arXiv: 1804.07461.

[21] Jacob Devlin, Ming Wei Chang, Kenton Lee και Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs]*, 2019. arXiv: 1810.04805.

[22] Corby Rosset.et al. *Turing-NLG: A 17-billion-parameter language model by Microsoft.* 2020.

[23] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever και Dario Amodei. *Language Models are Few-Shot Learners.* 2020.

[24] Yi Luan, Luheng He, Mari Ostendorf και Hannaneh Hajishirzi. *Multi-Task Identification of Entities, Relations, and Coreferencefor Scientific Knowledge Graph Construction. Proc. Conf. Empirical Methods Natural Language Process. (EMNLP)*, 2018.

[25] Franck Dernoncourt και Ji Young Lee. *PubMed 200k RCT: a Dataset for Sequential Sentence Classification in Medical Abstracts. Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, σελίδες 308–313, Taipei, Taiwan, 2017. Asian Federation of Natural Language Processing.

[26] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed και Michael Auli. *wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations*, 2020.

[27] Kevin Clark, Urvashi Khandelwal, Omer Levy και Christopher D. Manning. *What Does BERT Look At? An Analysis of BERT's Attention. arXiv:1906.04341 [cs]*, 2019. arXiv: 1906.04341.

[28] Mitchell A. Gordon, Kevin Duh και Nicholas Andrews. *Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning*, 2020.

[29] Victor Sanh, Thomas Wolf και Alexander M. Rush. *Movement Pruning: Adaptive Sparsity by Fine-Tuning. arXiv:2005.07683 [cs]*, 2020. arXiv: 2005.07683.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser και Illia Polosukhin. *Attention Is All You Need*, 2017.

[31] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrickvon Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest και Alexander M. Rush. *Transformers: State-of-the-Art Natural Language Processing. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, σελίδες 38–45, Online, 2020. Association for Computational Linguistics.

[32] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey και Noah A. Smith. *Don't Stop Pretraining: Adapt Language Models to Domains and Tasks*, 2020.

[33] Iz Beltagy, Kyle Lo και Arman Cohan. *SciBERT: A Pretrained Language Model for Scientific Text. EMNLP*. Association for Computational Linguistics, 2019.

[34] Emma Strubell, Ananya Ganesh και Andrew McCallum. *Energy and Policy Considerations for Deep Learning in NLP*, 2019.

[35] Roy Schwartz, Jesse Dodge, Noah A. Smith και Oren Etzioni. *Green AI*, 2019.

[36] Matthias Aßenmacher και Christian Heumann. *On the comparability of Pre-trained Language Models*, 2020.

[37] Anna Rogers, Olga Kovaleva και Anna Rumshisky. *A Primer in BERTology: What we know about how BERT works*, 2020.

[38] Wei Tsung Kao, Tsung Han Wu, Po Han Chi, Chun Cheng Hsieh και Hung Yi Lee. *BERT's output layer recognizes all hidden layers? Some Intriguing Phenomena and a simple way to boost BERT*, 2021.

[39] Joris Baan, Maartjeter Hoeve, Marliesvan der Wees, Anne Schuth και Maartende Rijke. *Understanding Multi-Head Attention in Abstractive Summarization*, 2019.

[40] Ian Tenney, Dipanjan Das και Ellie Pavlick. *BERT Rediscovers the Classical NLP Pipeline*, 2019.

[41] Nelson F. Liu, Matt Gardner, Yonatan Belinkov, Matthew E. Peters και Noah A. Smith. *Linguistic Knowledge and Transferability of Contextual Representations*, 2019.

[42] Adam Roberts, Colin Raffel και Noam Shazeer. *How Much Knowledge Can You Pack Into the Parameters of a Language Model?*, 2020.

[43] Yoav Goldberg. *Assessing BERT's Syntactic Abilities*, 2019.

[44] Yongjie Lin, Yi Chern Tan και Robert Frank. *Open Sesame: Getting Inside BERT's Linguistic Knowledge*, 2019.

[45] Geoffrey Hinton, Oriol Vinyals και Jeff Dean. *Distilling the Knowledge in a Neural Network*, 2015.

[46] Yann LeCun, John Denker και Sara Solla. *Optimal Brain Damage. Advances in Neural Information Processing Systems*D. Touretzky, επιμελητής, τόμος 2. Morgan-Kaufmann, 1990.

[47] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Martvan Baalen και Tijmen Blankevoort. *A White Paper on Neural Network Quantization*, 2021.

[48] Zhuliang Yao, Shijie Cao, Wencong Xiao, Chen Zhang και Lanshun Nie. *Balanced Sparsity for Efficient DNN Inference on GPU. Proceedings of the AAAI Conference on Artificial Intelligence*, 33:5676–5683, 2019.

[49] Yijun Yuan, Jiawei Hou, Andreas Nüchter και Sören Schwertfeger. *Self-supervised Point Set Local Descriptors for Point Cloud Registration*, 2020.

[50] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong και Qing He. *A Comprehensive Survey on Transfer Learning*, 2020.

[51] Sepp Hochreiter και Jürgen Schmidhuber. *Long Short-Term Memory. Neural Computation*, 9(8):1735–1780, 1997.

[52] Dzmitry Bahdanau, Kyunghyun Cho και Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*, 2016.

[53] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel και Yoshua Bengio. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, 2016.

[54] Elizbar Nadaraya. *On Estimating Regression. Theory of Probability and Its Applications*, 9:141–142, 1964.

[55] Kyunghyun Cho, Bartvan Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk και Yoshua Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, 2014.

[56] Minh Thang Luong, Hieu Pham και Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*, 2015.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren και Jian Sun. *Deep Residual Learning for Image Recognition*, 2015.

[58] Jimmy Lei Ba, Jamie Ryan Kiros και Geoffrey E. Hinton. *Layer Normalization*, 2016.

[59] Triet H. M. Le, Hao Chen και Muhammad Ali Babar. *Deep Learning for Source Code Modeling and Generation. ACM Computing Surveys*, 53(3):1–38, 2020.

[60] H. P. Luhn. *A Statistical Approach to Mechanized Encoding and Searching of Literary Information.* `https://doi.org/10.1147/rd.14.0309`, 1957.

[61] Karen Spärck Jones. *A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation*, 28:11–21, 1972.

[62] U. Fano. *Effects of Configuration Interaction on Intensities and Phase Shifts. Phys. Rev.*, 124:1866–1878, 1961.

[63] Kenneth Ward Church και Patrick Hanks. *Word Association Norms, Mutual Information, and Lexicography. 27th Annual Meeting of the Association for Computational Linguistics*, σελίδες 76–83, Vancouver, British Columbia, Canada, 1989. Association for Computational Linguistics.

[64] T. Mikolov. *Language Modeling for Speech Recognition in Czech.* 2007.

[65] Tomas Mikolov, Jiri Kopecky, Lukas Burget, Ondrej Glembek και Jan ?Cernocky. *Neural network based language models for highly inflective languages. 2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, σελίδες 4725–4728, 2009.

[66] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký και Sanjeev Khudanpur. *Recurrent Neural Network Based Language Model. Proceedings of the 11th Annual Conference of the International Speech Communication Association*, INTERSPEECH 2010, σελίδες 1045–1048. ISCA, 2010.

[67] Jeffrey Pennington, Richard Socher και Christopher Manning. *GloVe: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, σελίδες 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics.

[68] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee και Luke Zettlemoyer. *Deep contextualized word representations*, 2018.

[69] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei και Ilya Sutskever. *Language Models are Unsupervised Multitask Learners.* 2019.

[70] Alex Warstadt, Amanpreet Singh και Samuel R Bowman. *Neural Network Acceptability Judgments. arXiv preprint arXiv:1805.12471*, 2018.

[71] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng και Christopher Potts. *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, σελίδες 1631–1642, Seattle, Washington, USA, 2013. Association for Computational Linguistics.

[72] William B. Dolan και Chris Brockett. *Automatically Constructing a Corpus of Sentential Paraphrases. Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[73] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio και Lucia Specia. *SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, σελίδες 1–14, Vancouver, Canada, 2017. Association for Computational Linguistics.

[74] Adina Williams, Nikita Nangia και Samuel Bowman. *A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, σελίδες 1112–1122. Association for Computational Linguistics, 2018.

[75] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev και Percy Liang. *SQuAD: 100,000+ Questions for Machine Comprehension of Text. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, σελίδες 2383–2392, Austin, Texas, 2016. Association for Computational Linguistics.

[76] Ernest Davis Hector J Levesque και Leora Morgenstern. *The Winograd schema challenge. In AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning.* 2011.

[77] Song Han, Jeff Pool, John Tran και William J. Dally. *Learning both Weights and Connections for Efficient Neural Networks*, 2015.

[78] Qiangui Huang, Kevin Zhou, Suya You και Ulrich Neumann. *Learning to Prune Filters in Convolutional Neural Networks*, 2018.

[79] Ari S. Morcos, Haonan Yu, Michela Paganini και Yuandong Tian. *One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers*, 2019.

[80] Haonan Yu, Sergey Edunov, Yuandong Tian και Ari S. Morcos. *Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP*, 2020.

[81] Y. Lecun, L. Bottou, Y. Bengio και P. Haffner. *Gradient-based learning applied to document recognition. Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[82] Collin F. Baker, Charles J. Fillmore και John B. Lowe. *The Berkeley FrameNet Project. 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, σελίδες 86–90, Montreal, Quebec, Canada, 1998. Association for Computational Linguistics.

[83] Yanzhuo Ding, Yang Liu, Huanbo Luan και Maosong Sun. *Visualizing and Understanding Neural Machine Translation. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, σελίδες 1150–1159, Vancouver, Canada, 2017. Association for Computational Linguistics.

[84] Christos Louizos, Max Welling και Diederik P. Kingma. *Learning Sparse Neural Networks through L_0 Regularization*, 2018.

[85] Karen Simonyan και Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*, 2015.

[86] Marco Tulio Ribeiro, Sameer Singh και Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*, 2016.

List of Abbreviations

AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations from Transformers
CoLA	Corpus of Linguistic Acceptability
DL	Deep Learning
ELMo	Embeddings from Language Models
GLUE	General Language Understanding Evaluation
IP	Iterative Pruning
IMP	Iterative Magnitude Pruning
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
LSTM	Long short-term memory
LTH	Lottery Ticket Hypothesis
ML	Machine Learning
NMT	Neural Machine Translation
MLP	Multilayer Perceptron
MNLI	The Multi-Genre Natural Language Inference
MRPC	The Microsoft Research Paraphrase Corpus
NAS	Neural Architecture Search
NLI	Natural Language Inference
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
NNLM	Neural Network Language Model
QA	Question Answering
QNLI	Question-answering Natural Language Inference
QQP	Quora Question Pairs
RNN	Recurrent Neural Network
RTE	Recognizing Textual Entailment
SST-2	The Stanford Sentiment Treebank
STS-B	Semantic Textual Similarity Benchmark
WNLI	Winograd Natural Language Inference
XAI	Explainable Artificial Intelligence