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

Integrated Gradients for Structured Pruning on BERT

Diploma Thesis

of

DIMITRIOS BEKRIS

Supervisor: Alexandros Potamianos
Associate Professor, ECE NTUA

Athens, September 2024

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

Integrated Gradients for Structured Pruning on BERT

Diploma Thesis

of

DIMITRIOS BEKRIS

Supervisor: Alexandros Potamianos
Associate Professor, ECE NTUA

Approved by the examination committee on 13 September 2024.

(Signature)	*(Signature)*	*(Signature)*
Alexandros Potamianos	Petros Maragos	Athanasios Rontogiannis
Associate Professor, ECE NTUA	Professor, ECE NTUA	Associate Professor, ECE NTUA

Athens, September 2024

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

(Signature)

.....
Dimitrios Bekris
Electrical & Computer Engineer Graduate, NTUA

To humanity

Περίληψη

Τα μοντέλα Transformer έχουν φέρει επανάσταση στον τομέα της Επεξεργασίας Φυσικής Γλώσσας (NLP), ιδιαίτερα σε εργασίες όπως η ταξινόμηση κειμένου. Αυτά τα μοντέλα βασίζονται σε μεγάλο βαθμό στους μηχανισμούς προσοχής, που επιτρέπουν στο μοντέλο να εστιάζει σε διάφορα σημεία της εισόδου, βελτιώνοντας την κατανόηση του πλαισίου. Ωστόσο, υπάρχει μια συνεχιζόμενη συζήτηση σχετικά με το αν οι μηχανισμοί προσοχής μπορούν να θεωρηθούν αξιόπιστες εξηγήσεις για τις αποφάσεις του μοντέλου, όπως επισημαίνεται στη διαμάχη "Attention is not Explanation". Αυτή η διπλωματική εργασία εξετάζει τη δομημένη κλάδευση των κεφαλών προσοχής ως μέθοδο βελτιστοποίησης των μοντέλων Transformer, όπως το BERT, σε εργασίες ταξινόμησης του GLUE benchmark, μειώνοντας την πολυπλοκότητα του μοντέλου ενώ διατηρείται η απόδοση και η ερμηνευσιμότητα.

Το κίνητρο για αυτή την έρευνα πηγάζει από τη διαμάχη για τη δυνατότητα εξήγησης των μηχανισμών προσοχής. Εισάγεται ένας νέος δείκτης βασισμένος στη συσχέτιση που αξιοποιεί τη σχέση μεταξύ των τιμών προσοχής και των αποδόσεων, με στόχο την ταυτοποίηση των πιο σημαντικών κεφαλών προσοχής. Η προτεινόμενη μέθοδος βασίζεται στη Θεωρία του Τυχερού Δελτίου (Lottery Ticket Hypothesis) και δοκιμάζει τον αλγόριθμο Iterative Structured Pruning, ο οποίος προτάθηκε από τον Αχλατή. Αυτή η προσέγγιση στοχεύει να αξιολογήσει αν οι κεφαλές προσοχής που κλαδεύονται με βάση αυτόν τον δείκτη μπορούν να διατηρήσουν την απόδοση και την εξηγητική αξία του μοντέλου.

Η μέθοδος εφαρμόστηκε στο BERT και διεξήχθησαν εκτεταμένα πειράματα σε διάφορες εργασίες ταξινόμησης του GLUE Benchmark. Τα αποτελέσματα δείχνουν ότι η απόδοση είναι συγκρίσιμη με τη δουλειά του Αχλατή, με την προτεινόμενη προσέγγιση να επιτυγχάνει ανταγωνιστικά αποτελέσματα όσον αφορά την ακρίβεια και την αποδοτικότητα του μοντέλου.

Οι συνεισφορές αυτής της έρευνας εντοπίζονται τόσο στον τομέα του Structured Pruning όσο και στη συνεχιζόμενη συζήτηση για το αν οι μηχανισμοί προσοχής μπορούν να λειτουργήσουν ως εξηγήσεις. Με την ανάπτυξη και αξιολόγηση αυτής της μεθόδου, συμβάλλουμε στη βελτιστοποίηση των μοντέλων και προσφέρουμε νέες προοπτικές για την κατανόηση και αξιοποίηση των μηχανισμών προσοχής σε εργασίες NLP.

Λέξεις Κλειδιά

Βαθιά Μάθηση, Επεξεργασία Φυσικής Γλώσσας, Τρανσφερ Λεαρνινγ, Τρανσφορμερς, BERT, Συμπίεση, Δομική Περικοπή, Υπόθεσης Τυχερού Δελτίου, μετεκπαίδευση, Ερμηνευσιμότητα, Ολοκληρωμενες Κλίσεις Integrated Gradients

Abstract

Transformer models have revolutionized the field of Natural Language Processing (NLP), particularly in tasks like text classification. These models rely heavily on attention mechanisms to focus on different parts of the input, enhancing contextual understanding. However, there has been a growing debate on whether attention mechanisms can be considered reliable explanations for model decisions, as highlighted in the "Attention is not Explanation" debate. This thesis investigates structured pruning of attention heads as a method to optimize Transformer models like BERT for classification tasks in the GLUE benchmark, reducing model complexity while preserving performance and interpretability.

Motivated by the debate on the explanatory power of attention mechanisms, this research introduces a novel correlation-based metric that leverages the relationship between attention scores and attributions to identify the most important attention heads. The proposed method builds on the Lottery Ticket Hypothesis and tests the Iterative Structured Pruning algorithm, first introduced by Achlatis. This approach aims to assess whether attention heads that are pruned using this correlation-based metric can still maintain the model's performance and explanatory value.

The method was applied to BERT, and extensive experiments were conducted on various GLUE benchmark classification tasks. The results demonstrate that the performance is comparable to the work of Achlatis, with the proposed approach achieving competitive results in terms of both accuracy and model efficiency.

The contributions of this research lie in the Structured Pruning field as well as in the ongoing debate about whether attention can serve as an explanation. By developing and evaluating this method, we contribute to model optimization and offer new insights into how attention mechanisms can be understood and utilized in NLP tasks.

Keywords

Deep Learning, Natural Language Processing, Transfer Learning, Transformers, BERT, Compression, Structured Pruning, Lottery Ticket Hypothesis, fine-tuning, Interpretability / Explainability, Integrated Gradients

Ευχαριστίες

Η παρούσα διπλωματική εργασία αποτελεί ένα προσωπικό πόνημα στη διαμόρφωση του οποίου συνετέλεσαν πολλοί. Θα ήθελα να ευχαριστήσω πρωτίστως τον επιβλέποντα καθηγητή της εργασίας, τον καθηγητή Αλέξανδρο Ποταμιάνο για την ενασχόλησή του με την ερευνητική μου προσπάθεια και τις καίριες συμβουλές. Ένα μεγάλο ευχαριστώ οφείλω στον Γιώργο Παρασκευόπουλο για την πολύτιμη βοήθειά του και τις καθοριστικές του ιδέες. Επίσης, θα ήθελα να ευχαριστήσω τον Παναγιώτη Φιλντίση και Γιώργο Ρετσινά για την παραχώρηση πόρων του CVSP Lab (Computer Vision and Signal Processing). Τέλος, το μεγαλύτερο ευχαριστώ σε φίλους και γονείς που χωρίς τη στήριξή τους δεν θα μπορούσα να τη φέρω εις πέρας.

Table of Contents

List of Figures

List of Tables

Εκτεταμένη Περίληψη στα Ελληνικά

Στο κεφάλαιο αυτό παρουσιάζεται μία εκτεταμένη περίληψη της εργασίας αυτής στα ελληνικά. Με συνοπτικό τρόπο θα διατυπωθούν οι κεντρικές ιδέες από κάθε ενότητα.

0.1 Περίληψη

Η έρευνα αυτή εξερευνά και βελτιστοποιεί τους μηχανισμούς προσοχής στα μοντέλα Transformer, τα οποία αποτελούν κλειδί στην Επεξεργασία Φυσικής Γλώσσας (NLP) και σε διάφορους τομείς της μηχανικής μάθησης. Το επίκεντρο της έρευνας είναι η δομημένη κλάδευση των κεφαλών προσοχής, μια εξελιγμένη τεχνική που στοχεύει στη βελτιστοποίηση των μοντέλων μέσω της επιλεκτικής αφαίρεσης κεφαλών προσοχής, διατηρώντας και ενισχύοντας την απόδοση του μοντέλου, ενώ ταυτόχρονα μειώνει την υπολογιστική πολυπλοκότητα και τις απαιτήσεις πόρων. Η μελέτη αυτή εισάγει μια νέα προσέγγιση που αξιοποιεί την τεχνική Neuron Conductance [16] για τη μέτρηση της σημασίας των νευρώνων, εξετάζοντας τη συσχέτιση μεταξύ των βαθμολογιών προσοχής και των αντίστοιχων αποδόσεών τους. Αυτή η προσέγγιση παρέχει μια λεπτομερή κατανόηση της αλληλεπίδρασης μεταξύ των μηχανισμών προσοχής και της εγγενούς σημασίας των ατομικών κεφαλών προσοχής. Ένας νέος δείκτης, εμπνευσμένος από προηγούμενες εργασίες, αναπτύσσεται για τον καθορισμό της σημασίας των κεφαλών προσοχής, διευκολύνοντας πιο ενημερωμένη και αποτελεσματική κλάδευση. Επιπλέον, η μέθοδος αυτή μελετάται μέσα από το πρίσμα της Υπόθεσης του Λαχείου, δείχνοντας ότι ο δείκτης μας ανταγωνίζεται τις προσεγγίσεις των Michel et al. [17] και Achlatis [18], αποδίδοντας αποτελέσματα που είναι παρόμοια με εκείνα που προκύπτουν σε μεγάλα σύνολα δεδομένων του GLUE. Επιπλέον, διεξάγουμε πειράματα για τη μέθοδο που προτάθηκε για πρώτη φορά από τον Achlatis στο έργο του, με τίτλο "Iteratively Structured Pruning". Οι μεθοδολογίες και οι επιπτώσεις αυτής της έρευνας συμβάλλουν όχι μόνο με μια νέα οπτική στις τεχνικές βελτιστοποίησης μοντέλων, αλλά επίσης στοχεύουν στην προώθηση της κατανόησης της δομημένης κλάδευσης και της επίδρασής της στην απόδοση του μοντέλου, οδηγώντας ενδεχομένως στην ανάπτυξη πιο ανθεκτικών και αποδοτικών μοντέλων.

0.2 Εισαγωγή

Οι μηχανισμοί προσοχής, ιδιαίτερα εκείνοι που χρησιμοποιούνται στα μοντέλα Transformer [4], έχουν καταστεί θεμελιώδεις στην NLP και σε διάφορους άλλους τομείς της μηχανικής μάθησης. Αυτοί οι μηχανισμοί επιτρέπουν στα μοντέλα να ζυγίζουν και να δίνουν προτεραιότητα σε διάφορα τμήματα της εισόδου κατά τη δημιουργία εξόδου, επιτρέποντας πιο λεπτομερείς και προσαρμοσμένες στην περίσταση αναπαραστάσεις των πληροφοριών. Μέσα

σε αυτούς τους μηχανισμούς, οι κεφαλές προσοχής είναι κρίσιμες για τη δημιουργία ποικίλων γραμμικών μετασχηματισμών της εισόδου, επιτρέποντας στο μοντέλο να εστιάζει σε διαφορετικές πτυχές ή χαρακτηριστικά των δεδομένων εισόδου [7].

Ένα από τα πιο γνωστά μοντέλα Transformer είναι το BERT, που προτάθηκε από τον Devlin et al. [7]. Το BERT είναι ένα προεκπαιδευμένο μοντέλο αναπαράστασης γλώσσας που εκπαιδεύεται από αμείλικτο κείμενο, συνδυάζοντας ταυτόχρονα το αριστερό και το δεξιό πλαίσιο σε όλα τα επίπεδα. Ως αποτέλεσμα, το προεκπαιδευμένο μοντέλο BERT μπορεί να βελτιστοποιηθεί με την προσθήκη μόνο μιας επιπλέον στρώσης εξόδου για τη δημιουργία μοντέλων τελευταίας τεχνολογίας για ένα ευρύ φάσμα εργασιών, όπως η απάντηση σε ερωτήματα και η γλωσσική συνεπαγωγή, χωρίς σημαντικές αλλαγές στην αρχιτεκτονική της εργασίας.

Τα μοντέλα που βασίζονται σε Transformer περιέχουν εκατομμύρια παραμέτρους, οι οποίες επιβραδύνουν την εξαγωγή συμπερασμάτων, αυξάνουν το αποτύπωμα μνήμης, τον αριθμό των υπολογιστικών πράξεων (FLOPs), τη χρήση ενέργειας και συμβάλλουν στα περιβαλλοντικά ζητήματα. Αυτό το πρόβλημα τείνει να κλιμακωθεί γρήγορα. Για παράδειγμα, το BERT-base [7] περιέχει 110 εκατομμύρια παραμέτρους, ενώ μοντέλα όπως το GPT-4 [19], το Gemini της Google [20] και το Claude της Anthropic [21] περιέχουν εκατοντάδες δισεκατομμύρια παραμέτρους.

Μία από τις απαντήσεις σε αυτό το πρόβλημα είναι η συμπίεση του μοντέλου. Οι ερευνητές έχουν εφαρμόσει τεχνικές κλάδευσης στα μοντέλα BERT-base, κλαδεύοντας βάρη ή δομημένα στοιχεία, όπως οι κεφαλές προσοχής. Ερευνητές όπως οι Voita et al. [11], Kovaleva et al. [10] και Michel et al. [17] προτείνουν ότι τα μοντέλα που βασίζονται σε Transformer είναι βαριά υπερπαραμετροποιημένα, επιτρέποντας την αφαίρεση ενός μεγάλου αριθμού κεφαλών χωρίς σημαντική μείωση της απόδοσης.

Αυτή η έρευνα στοχεύει να εμβαθύνει στις μεθοδολογίες και τις επιπτώσεις της δομημένης κλάδευσης των κεφαλών προσοχής, με μια νέα προσέγγιση που επικεντρώνεται σε μια παραλλαγή των Integrated Gradients [22] για τη μέτρηση της σημασίας των κεφαλών προσοχής. Τα Integrated Gradients έχουν αναδειχθεί ως μια σημαντική μέθοδος για τη σημασία των χαρακτηριστικών στα μοντέλα βαθιάς μάθησης, παρέχοντας πληροφορίες για τις αποφάσεις των μοντέλων [23]. Αυτή η έρευνα αξιοποιεί την τεχνική Neuron Conductance [24], μια μέθοδο για τη μέτρηση της σημασίας των νευρώνων σε σχέση με την έξοδο του μοντέλου, με τις εξής συνεισφορές:

- Εξέταση του πίνακα συσχέτισης μεταξύ των βαθμολογιών προσοχής και των αντίστοιχων αποδόσεών τους, προσφέροντας μια λεπτομερή κατανόηση της αλληλεπίδρασης μεταξύ των μηχανισμών προσοχής και της εγγενούς σημασίας των ατομικών κεφαλών προσοχής.

- Ανάπτυξη ενός νέου δείκτη βασισμένου στη συσχέτιση μεταξύ των αποδόσεων Neuron Conductance και των βαθμολογιών προσοχής, διευκολύνοντας πιο ενημερωμένη και αποτελεσματική κλάδευση των λιγότερο σημαντικών κεφαλών.

- Αντλώντας έμπνευση από προηγούμενες εργασίες [17, 25], αυτή η έρευνα εισάγει μια βελτιωμένη μέθοδο για τον καθορισμό της σημασίας των κεφαλών προσοχής, ενισχύοντας την αποδοτικότητα και αποτελεσματικότητα των μεγάλων γλωσσικών μοντέλων.

Αυτή η καινοτόμος προσέγγιση βασίζεται στη διπλωματική εργασία του Achlatis [18], προσφέροντας μια φρέσκια οπτική στη συζήτηση για τις τεχνικές βελτιστοποίησης μοντέλων. Συνδυάζοντας τις πληροφορίες από τη συσχέτιση μεταξύ των βαθμολογιών προσοχής και των αποδόσεων, αυτή η έρευνα στοχεύει να προωθήσει την κατανόηση της δομημένης κλάδευσης και της επίδρασής της στην απόδοση του μοντέλου, ανοίγοντας τον δρόμο για την ανάπτυξη πιο ανθεκτικών και αποδοτικών μοντέλων.

0.3 Σχετική Εργασία

Η κλάδευση στα μοντέλα Transformer, ιδιαίτερα στις αρχιτεκτονικές όπως το BERT, έχει μελετηθεί εκτενώς για να βελτιώσει την απόδοση των μοντέλων χωρίς σημαντική απώλεια στην απόδοση. Μια θεμελιώδης μελέτη από τους Michel et al. (2019) [17] έδειξε ότι πολλές κεφαλές προσοχής στα Transformers είναι περιττές, αμφισβητώντας την άποψη ότι όλες οι κεφαλές είναι εξίσου σημαντικές. Η προσέγγισή τους για δομημένη κλάδευση αφαιρεί διαδοχικά τις λιγότερο σημαντικές κεφαλές βάσει ενός δείκτη σημασίας που υπολογίζεται σε όλο το σύνολο εκπαίδευσης, αποκαλύπτοντας ότι η σημασία των κεφαλών προσοχής καθορίζεται νωρίς στην εκπαίδευση και παραμένει σταθερή.

Στη συνέχεια, οι Voita et al. (2019) [11] πρότειναν μια μέθοδο για δομημένη κλάδευση που χρησιμοποιεί μια ευριστική συνάρτηση βασισμένη σε μια κατανομή Hard Concrete για να διατηρήσει επιλεκτικά μόνο τις πιο κρίσιμες κεφαλές. Τα ευρήματά τους συμφωνούν με εκείνα των Michel et al. [17], ενισχύοντας την ιδέα ότι ένα σημαντικό ποσοστό των κεφαλών προσοχής μπορεί να κλαδευτεί χωρίς να μειωθεί η απόδοση του μοντέλου.

Η έννοια της κλάδευσης έχει επίσης εξεταστεί από την προοπτική της Υπόθεσης του Λαχείου (Lottery Ticket Hypothesis - LTH), που εισήχθη από τους Frankle και Carbin (2019) [9], η οποία προτείνει ότι μέσα σε ένα πυκνό νευρωνικό δίκτυο υπάρχει ένα αραιό υποδίκτυο ικανό να ανταγωνιστεί την απόδοση του πλήρους μοντέλου. Ενώ οι Frankle και Carbin επικεντρώθηκαν σε μη δομημένη κλάδευση σε μικρότερα δίκτυα, επόμενες εργασίες όπως αυτές των Liu et al. (2019) [26] και Chen et al. (2020) [13] δοκίμασαν την LTH σε μεγαλύτερα μοντέλα και πιο περίπλοκες ρυθμίσεις, συμπεριλαμβανομένων προεκπαιδευμένων δικτύων BERT. Οι Liu et al. (2019) [26] επικρίνουν τη γενικευσιμότητα της LTH, υποστηρίζοντας ότι η δομημένη κλάδευση σε μεγάλα μοντέλα, ιδιαίτερα υπό διαφορετικές ρυθμίσεις βελτιστοποίησης, μπορεί να αποφέρει διαφορετικά αποτελέσματα από αυτά που παρατηρούνται σε μικρότερες, μη δομημένες ρυθμίσεις. Από την άλλη πλευρά, οι Chen et al. (2020) [13] επέκτειναν την LTH στο BERT, δείχνοντας ότι τα υποδίκτυα που εντοπίστηκαν μέσω της κλάδευσης μεγέθους μπορούν να αποδώσουν εξίσου με το πλήρες μοντέλο, ιδιαίτερα όταν αρχικοποιούνται με προεκπαιδευμένα βάρη.

Εξερευνώντας περαιτέρω την αλληλεπίδραση μεταξύ της κλάδευσης και της ερμηνευσιμότητας των μοντέλων, οι Prasanna et al. (2020) [12] και Yang et al. (2021) [27] διερεύνησαν την αποτελεσματικότητα της δομημένης κλάδευσης στην κατανόηση της συμπεριφοράς των μοντέλων. Οι Prasanna et al. [12] διαπίστωσαν ότι η κλάδευση βάσει δεικτών σημασίας, ιδιαίτερα στο BERT, μπορεί να αποκαλύψει ποια κεφαλές και MLPs είναι πιο κρίσιμα για την απόδοση των εργασιών, με τις κεφαλές στο μέσο στρώμα να είναι συχνά πιο μεταφέρσιμες μεταξύ των εργασιών. Οι Yang et al. επέκτειναν αυτό χρησιμοποιώντας μεθόδους βασισ-

μένες στην απόδοση για να καθοδηγήσουν τις αποφάσεις κλάδευσης, υπογραμμίζοντας τη δυνατότητα συμπίεσης προσαρμοσμένης στις εργασίες σε μοντέλα πολλαπλών εργασιών.

Οι Hao et al. (2021) [28] εισήγαγαν μια νέα μέθοδο ερμηνευσιμότητας που ονομάζεται Απόδοση Αυτοπροσοχής (Self-Attention Attribution - AttAttr) που αξιοποιεί τις αποδόσεις για την κλάδευση των Transformers. Η προσέγγισή τους επικεντρώνεται στις συνδέσεις των κεφαλών προσοχής, μετατρέποντας τις αποδόσεις σε έναν δείκτη κλάδευσης λαμβάνοντας το μέγιστο απόδοση ανά σύνδεση και υπολογίζοντας τον μέσο όρο σε όλα τα δείγματα. Αυτή η μέθοδος μοιράζεται εννοιολογικές ομοιότητες με τα Integrated Gradients, αλλά σχεδιάστηκε ειδικά για να στοχεύει κεφαλές προσοχής. Σε αντίθεση με τις μεθόδους που βασίζονται στην Υπόθεση του Λαχείου, η προσέγγιση των Hao κλαδεύει διαδοχικά τις κεφαλές προσοχής με μάσκες, αντί να τις αφαιρεί μόνιμα, και αξιολογεί τα κλαδεμένα μοντέλα χωρίς να επαναρχικοποιεί τα βάρη. Αυτή η τεχνική υπογραμμίζει μια κρίσιμη διαφορά από την προσέγγισή μου, όπου η συσχέτιση μεταξύ των βαθμολογιών προσοχής και των αποδόσεων υπολογίζεται για να καθοδηγήσει την κλάδευση, με τις αποδόσεις να υπολογίζονται σε σχέση με τις προβλέψεις του μοντέλου και όχι με τις χρυσές ετικέτες, όπως στο έργο των Hao.

Οι πρόσφατες προόδους από τους Ilhan et al. (2024) [29] και Grover et al. (2023) [30] βελτίωσαν περαιτέρω τις τεχνικές κλάδευσης, εισάγοντας αντίστοιχα μεθόδους αποδοτικής χρήσης πόρων και κλάδευσης βασισμένης στην ερμηνευσιμότητα. Οι Ilhan et al. [29] πρότειναν μια ευριστική μέθοδο κλάδευσης που βελτιστοποιεί τη διαδικασία προσαρμογής ενημερώνοντας επιλεκτικά τα βάρη, προσφέροντας μια πρακτική λύση για την ανάπτυξη μοντέλων μεγάλης κλίμακας. Εν τω μεταξύ, οι Grover et al. [30] αντιμετώπισαν την πρόκληση των θορυβώδων κλίσεων στην κλάδευση, εισάγοντας μεθόδους που ενσωματώνουν άμεσα την πληροφορία της κλίσης στον δείκτη κλάδευσης, δείχνοντας ανώτερη απόδοση σε περιπτώσεις βαθιάς κλάδευσης.

Συνολικά, αυτές οι μελέτες αναδεικνύουν την εξελισσόμενη κατανόηση της κλάδευσης στα μοντέλα Transformer, με τις δομημένες και μη δομημένες μεθόδους να προσφέρουν συμπληρωματικές πληροφορίες. Από τον εντοπισμό περιττών στοιχείων μέσα στο BERT έως την αξιοποίηση της κλάδευσης για την ερμηνευσιμότητα και την αποδοτικότητα, η έρευνα υπογραμμίζει την κρίσιμη ισορροπία μεταξύ της πολυπλοκότητας του μοντέλου και της απόδοσης, καθοδηγώντας την ανάπτυξη πιο αποδοτικών μοντέλων NLP.

0.4 Ορισμός του Προβλήματος

0.4.1 Ορισμός Δομημένης Κλάδευσης

Δεδομένου ενός συνόλου δεδομένων $D = \{(x_i, y_i)\}_{i=1}^{n}$ και ενός επιθυμητού επιπέδου αραιότητας κ, η δομημένη κλάδευση ενός νευρωνικού δικτύου μπορεί να διατυπωθεί ως το παρακάτω πρόβλημα βελτιστοποίησης με περιορισμούς:

$$\begin{aligned} \min_{w_s} L(w_s; D) &= \min_{w_s} \frac{1}{n} \sum_{i=1}^{n} \ell(w_s; (x_i, y_i)) \\ \text{s.t.} \quad w_s &\in \mathbb{R}^m, \quad \|w_s\|_0 \leq \kappa \end{aligned} \tag{1}$$

Όπου $\ell(\cdot)$ είναι η συνήθης συνάρτηση απώλειας, w_s είναι το δομημένο σύνολο παραμέτρων του νευρωνικού δικτύου (π.χ. κεφαλές προσοχής), m είναι ο συνολικός αριθμός των δομημένων συνόλων και η $\|\cdot\|_0$ αναφέρεται στο L0 νόρμα.

Η ελαχιστοποίηση της L0 νόρμας είναι προκλητική, καθώς είναι μη κυρτή, NP-δύσκολη, και απαιτεί συνδυαστική αναζήτηση, καθιστώντας την δομημένη κλάδευση ένα NP-δύσκολο πρόβλημα. Η δομημένη κλάδευση μπορεί να εκτελεστεί πριν, κατά τη διάρκεια ή μετά την προσαρμογή του μοντέλου. Σ' αυτή την εργασία, πρώτα προσαρμόζουμε το μοντέλο και στη συνέχεια εφαρμόζουμε τη μέθοδο κλάδευσης μας.

0.4.2 Αρχιτεκτονική BERT

Το BERT αποτελείται από μια σειρά επιπέδων κωδικοποιητών Transformer [4]. Κάθε επίπεδο έχει μια ταυτόσημη δομή: ένα μπλοκ πολλαπλών κεφαλών αυτοπροσοχής (MHAtt) ακολουθούμενο από έναν πολυστρωματικό αντιληπτικό νευρωνικό δίκτυο (MLP), με συνδέσεις υπολοίπου γύρω από κάθε ένα.

Το μπλοκ MHAtt περιέχει N_h ανεξάρτητα παραμετροποιημένες κεφαλές. Μια κεφαλή προσοχής h στο επίπεδο l παραμετροποιείται από μήτρες $W_k^h, W_q^h, W_v^h \in \mathbb{R}^{d_h \times d}$ και $W_o^h \in \mathbb{R}^{d \times d_h}$, όπου d_h συνήθως ορίζεται σε d/N_h. Δεδομένων n d-διάστατων διανυσμάτων εισόδου $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^d$, το MHAtt υπολογίζεται ως το άθροισμα των εξόδων κάθε ατομικής κεφαλής που εφαρμόζεται στην είσοδο x:

$$\text{MHAtt}(x, q) = \sum_{h=1}^{N_h} \text{Att}(W_k^h, W_q^h, W_v^h, W_o^h)(x, q) \tag{2}$$

Για να επιτρέψει την αλληλεπίδραση διαφορετικών κεφαλών προσοχής μεταξύ τους, οι Transformers εφαρμόζουν ένα μη γραμμικό πολυεπίπεδο νευρωνικό δίκτυο πάνω από την έξοδο του MHAtt σε κάθε επίπεδο του Transformer [4]. Κάθε κεφαλή προσοχής μπορεί να εστιάσει σε διαφορετικές πτυχές της εισόδου, όπως συντακτικό και σημασιολογικό περιεχόμενο.

0.5 Προτεινόμενη Μέθοδος

0.5.1 Περιγραφή Μεθόδου

Πρώτον, όπως πρότειναν οι Michel et al. [17], εισάγουμε μεταβλητές μάσκας ξ_h με τιμές στο $\{0, 1\}$, όπου $\xi_h = 1$ σημαίνει ότι η αντίστοιχη κεφαλή h δεν είναι κρυμμένη, ενώ $\xi_h = 0$ σημαίνει ότι η κεφαλή h είναι κρυμμένη. Αυτό οδηγεί σε μια τροποποίηση της φόρμουλας για το Multi-Head Attention (MHAtt):

$$\text{MHAtt}(x, q) = \sum_{h=1}^{N_h} \xi_h \cdot \text{Att}_{W_k^h, W_q^h, W_v^h, W_o^h}(x, q) \tag{3}$$

Αυτή η τροποποίηση μας επιτρέπει να κρύψουμε τις κλαδεμένες κεφαλές χωρίς να τις αφαιρέσουμε πραγματικά.

Ορίζουμε έναν νέο δείκτη, βασισμένο στη συσχέτιση μεταξύ των βαθμολογιών προσοχής των τοκενών και των αντίστοιχων αποδόσεων:

$$I_{L,H} = \mathbb{E}_x \left(\text{corr} \left(\text{Attr}(A_{L,H}), A_{L,H} \right) \right) \tag{4}$$

όπου x είναι η κατανομή δεδομένων, $A_{L,H}$ η βαθμολογία προσοχής του επιπέδου L και της κεφαλής H, και $\text{Attr}(A_{L,H})$ η απόδοση αυτής της κεφαλής προσοχής. Τόσο το A όσο και το $\text{Attr}(A)$ έχουν τις ίδιες διαστάσεις με `num_tokens` × `num_tokens`.

Για την απόδοση, χρησιμοποιούμε το Neuron Conductance, που καθορίζει αν η σύνδεση προσοχής (i, j) έχει σημαντική επιρροή στην πρόβλεψη του μοντέλου. Εμπνευσμένοι από τους Hao et al. [25], χρησιμοποιούμε τη συσχέτιση αντί για τη συνάρτηση max για την συλλογή των πληροφοριών.

Ο στόχος εδώ είναι να αντιμετωπίσουμε κάθε κεφαλή προσοχής ως μια μεταβλητή x_i και κάθε απόδοση ως μια μεταβλητή y_i, με τις μετρήσεις κάθε μεταβλητής για τον υπολογισμό της συσχέτισης να είναι οι συνδέσεις των σχέσεων των τοκενών σε κάθε πίνακα.

Στην έρευνά μας, προτείνουμε έναν νέο δείκτη, που χρησιμοποιείται για τη Δομημένη Κλάδευση. Η συμβολή μας σε αυτό το πεδίο βασίζεται σε 2 κεντρικούς αλγορίθμους. Αυτές είναι οι κύριες ιδέες που αναφέρθηκαν στο έργο του Achlatis [18].

Αλγοριθμος 0.1: *Δομημένη Κλάδευση με Δείκτη Σημασίας*

Είσοδος: Πλήρως προσαρμοσμένο προεκπαιδευμένο δίκτυο
Ορίστε την αρχική μάσκα κλάδευσης των κεφαλών προσοχής σε $s = \mathbf{1}_d$ ▷ όπου d είναι η διάσταση
repeat
    Υπολογίστε το I_h για τις μη κλαδεμένες κεφαλές προσοχής
    Ταξινομήστε τις κεφαλές με φθίνουσα σειρά βάσει του I_h
    Κλαδέψτε το $\kappa\%$ των αρχικών κεφαλών με το χαμηλότερο I_h και ενημερώστε το s
until η αραιότητα του s φτάσει στο s_T ▷ s_T: Όριο Αραιότητας
Έξοδος: Μάσκα κλάδευσης s

Αλγοριθμος 0.2: *Επαναληπτική Δομημένη Κλάδευση (ISP)*

Είσοδος: Πλήρως προσαρμοσμένο προεκπαιδευμένο δίκτυο
Ορίστε την αρχική μάσκα κλάδευσης των κεφαλών προσοχής σε $s = \mathbf{1}_d$
repeat
    Υπολογίστε το I_h για τις μη κλαδεμένες κεφαλές προσοχής
    Ταξινομήστε τις κεφαλές με φθίνουσα σειρά βάσει του I_h
    Κλαδέψτε το $\kappa\%$ των εναπομείναντων κεφαλών με το χαμηλότερο I_h από το προεκπαιδευμένο μοντέλο και ενημερώστε το s
    Προσαρμόστε εκ νέου το προεκπαιδευμένο δίκτυο
until η αραιότητα του s φτάσει στο s_T ▷ s_T: Όριο Αραιότητας
Έξοδος: Μάσκα κλάδευσης s

Για τον Αλγόριθμο 6.1, χρησιμοποιούμε αυτό όπως προτάθηκε από τον Achlatis στη διπλω-

ματική του εργασία. Πρώτα, βρίσκουμε τις μάσκες και αξιολογούμε στο σύνολο ανάπτυξης και στη συνέχεια επεκτείνουμε την προσέγγιση και εφαρμόζουμε τις μάσκες στο προεκπαιδευμένο μοντέλο και το προσαρμόζουμε, προκειμένου να βρούμε τα νικητήρια εισιτήρια (Υπόθεση του Λαχείου). Η τελευταία αυτή μέθοδος εφαρμόστηκε επίσης από τους Prasanna et al. [12]. Την αποκαλούμε Integrated Gradients Structured Pruning Fine-Tuned (IGSPF).

Σε αυτό το σημείο, εμπνευσμένοι από την υλοποίηση των πειραμάτων του Michel, εφαρμόζουμε τις μάσκες στο προεκπαιδευμένο μοντέλο σε κάθε επανάληψη και το προσαρμόζουμε. Αυτό είναι παρόμοιο με τον αλγόριθμο ISP 6.2, αλλά σε αυτή την περίπτωση, σε κάθε επανάληψη κλαδεύουμε έναν συγκεκριμένο αριθμό κεφαλών από το αρχικό. Το ονομάζουμε Integrated Gradients Structured Pruning Pre-trained (IGSPP).

Όσον αφορά τον αλγόριθμο ISP 6.2, εφαρμόζουμε τον δείκτη μας στην ιδέα του Achlatis για να επεκτείνουμε τον αλγόριθμο IMP που πρότεινε ο Chen στο έργο του [13] για τη Δομημένη Κλάδευση. Η κύρια διαφορά εδώ από τον πρώτο αλγόριθμο είναι ότι σε κάθε επανάληψη ο λόγος κλαδέματος είναι σταθερός, αλλά ο αριθμός των κεφαλών δεν είναι. Αυτό βασίζεται στην Υπόθεση του Λαχείου ότι οι Επαναληπτικοί Αλγόριθμοι θα μπορούσαν να βρουν πιο συμπαγή και εύκολα μαθηματικά υποδίκτυα.

0.5.2 Η Έννοια Πίσω από τη Συσχέτιση

Ουσιαστικά, ο σκοπός είναι να διατηρήσουμε τις κεφαλές με μεγαλύτερη θετική μονοτονική σχέση μεταξύ των βαθμολογιών προσοχής και των αποδόσεων. Δηλαδή, η γενικότερη συμπεριφορά της κεφαλής προσοχής είναι ότι όταν μια σύνδεση προσοχής αυξάνεται, η συμβολή της στην πρόβλεψη αυξάνεται ταυτόχρονα.

Το κίνητρο

Στην έρευνά μου, επικεντρώθηκα στην πρακτική εφαρμογή της δομημένης κλάδευσης στα μοντέλα NLP, χρησιμοποιώντας έναν νέο δείκτη που συσχετίζει τους μηχανισμούς προσοχής με τις αποδόσεις που προκύπτουν από τη μέθοδο Neuron Conductance—μια εξελιγμένη εκδοχή των Integrated Gradients. Αυτή η εργασία είχε ως κίνητρο την συνεχιζόμενη συζήτηση για το αν οι μηχανισμοί προσοχής μπορούν να χρησιμεύσουν ως αξιόπιστες εξηγήσεις για τη συμπεριφορά των μοντέλων, όπως τονίστηκε στα άρθρα "Attention is Not Explanation" [31] και "Attention is Not Not Explanation" [32].

Το κύριο εύρημα από την έρευνά μου είναι ότι διασφαλίζοντας ότι τα βάρη της προσοχής ευθυγραμμίζονται με αξιόπιστες μεθόδους απόδοσης, μπορούμε να διατηρήσουμε σημαντικά ερμηνευτικά χαρακτηριστικά ενός μοντέλου ακόμα και μετά από σημαντική κλάδευση. Αυτό όχι μόνο αμφισβητεί την άποψη ότι οι μηχανισμοί προσοχής δεν είναι αξιόπιστοι ως εξηγήσεις, αλλά προσφέρει και ένα πρακτικό πλαίσιο για τη βελτιστοποίηση μοντέλων που διατηρεί υψηλό βαθμό ερμηνευσιμότητας. Η εργασία μου συμβάλλει στη συζήτηση δείχνοντας ότι η προσοχή, όταν ενσωματώνεται προσεκτικά με μεθόδους απόδοσης, μπορεί να είναι ένα πολύτιμο εργαλείο τόσο για την κατανόηση όσο και για τη βελτίωση των μοντέλων NLP.

Η ιδέα εδώ είναι ότι οι προσοχές από μόνες τους δεν επαρκούν για να παρέχουν χρήσιμες πληροφορίες, οπότε συνδυάζουμε σταθερές αποδόσεις για να δούμε τι συμβαίνει, ώστε να συμβάλλουμε και στις δύο κατευθύνσεις: Δομημένη Κλάδευση και Προσοχές ως Εξήγηση.

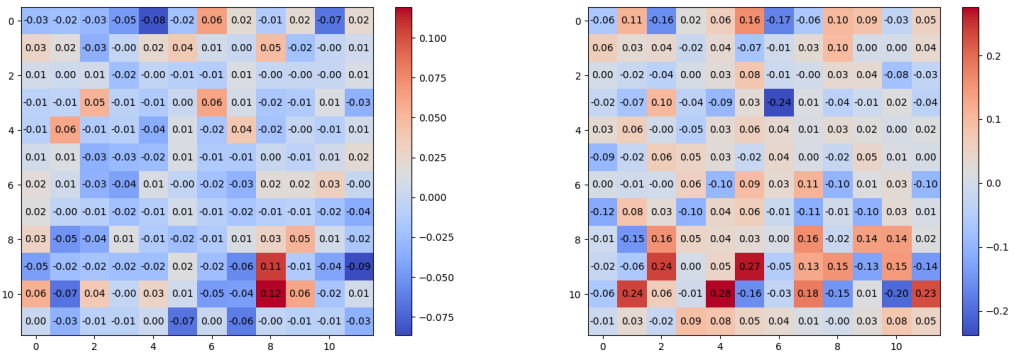
Spearman. Γιατί;

Ο στόχος εδώ είναι να διατηρήσουμε τις κεφαλές προσοχής των οποίων οι δύο μεταβλητές έχουν παρόμοια συμπεριφορά, δηλαδή παρόμοια ένταση και κατεύθυνση μεταξύ των συνδέσεων.

Γι' αυτό, επέλεξα τη Συσχέτιση Βαθμίδων Spearman αντί για το Kendall's Tau και τη συσχέτιση Pearson λόγω της καταλληλότητάς της για μη γραμμικές σχέσεις και της ευαισθησίας της σε διαφορές βαθμίδων. Η συσχέτιση Pearson προϋποθέτει γραμμικότητα και είναι ευαίσθητη σε εξωγενείς τιμές, κάνοντάς την λιγότερο κατάλληλη για βαθμολογίες προσοχής και αποδόσεις, οι οποίες μπορεί να μην ακολουθούν γραμμικό πρότυπο. Η Συσχέτιση Βαθμίδων Spearman, ωστόσο, καταγράφει μονοτονικές σχέσεις και παρέχει μια ολοκληρωμένη αξιολόγηση της συνολικής συμφωνίας κατάταξης, καθιστώντας την πιο αποτελεσματική για την αξιολόγηση των γενικών τάσεων. Επιπλέον, η Spearman είναι πιο ευαίσθητη σε σημαντικές αποκλίσεις στη βαθμολογία, προσφέροντας μια πιο λεπτομερή κατανόηση των δεδομένων σε σχέση με το Kendall's Tau. Αυτό καθιστά τη Spearman την πιο κατάλληλη επιλογή για την ανάλυση της ευθυγράμμισης μεταξύ βαθμολογιών προσοχής και αποδόσεων σε αυτή τη μελέτη.

Ανάλυση Συσχέτισης

Αρχικά, αξιοποιούμε ένα μικρό υποσύνολο των συνόλων δεδομένων IMDB, RTT και SST-2, περιορίζοντας τις εισόδους μας σε το πολύ 200 τοκενς λόγω περιορισμών πόρων. Αυτό το υποσύνολο χρησιμοποιείται για τη διεξαγωγή προκαταρκτικής ανάλυσης των τιμών συσχέτισης τόσο στο προεκπαιδευμένο μοντέλο (εκπαιδευμένο στην εργασία MLM) όσο και στο προσαρμοσμένο μοντέλο προσαρμοσμένο στις αντίστοιχες εργασίες. Στο Σχήμα 6.1, παρουσιάζουμε τους πίνακες συσχέτισης τόσο για το προεκπαιδευμένο όσο και για το προσαρμοσμένο μοντέλο, όπου οι κλίσεις έχουν υπολογιστεί σε σχέση με την τελική πρόβλεψη για το SST-2. (Δείτε το παράρτημα .1 για περισσότερα σύνολα δεδομένων)



(a) Προεκπαιδευμένο Μοντέλο *(b) Προσαρμοσμένο Μοντέλο*

Figure 1. *Θερμοκάρτες για τη συσχέτιση του προεκπαιδευμένου και προσαρμοσμένου μοντέλου για το σύνολο δεδομένων SST-2.*

Όπως παρατηρείται στο Σχήμα 6.1, υπάρχει σημαντική διαφορά μεταξύ των τιμών συσχέτισης

στο προεκπαιδευμένο και το προσαρμοσμένο μοντέλο, υποδεικνύοντας ότι τα βάρη της προσοχής έχουν ενσωματώσει αποτελεσματικά γνώση ειδικής για την εργασία κατά τη διάρκεια της προσαρμογής. Επιπλέον, παρατηρούμε την παρουσία κεφαλών προσοχής με αρνητικές τιμές συσχέτισης, οι οποίες είναι ισχυροί υποψήφιοι για κλάδευση. Αν και οι συνολικές τιμές συσχέτισης δεν είναι εξαιρετικά υψηλές, υπάρχουν ακόμα πολύτιμες πληροφορίες που μπορούν να αντληθούν. Συγκεκριμένα, το 3.47% των κεφαλών προσοχής για το σύνολο δεδομένων IMDB έχει τιμές συσχέτισης μεταξύ 0.2 και 0.4, ενώ το 2.08% και το 4.86% των κεφαλών στα σύνολα δεδομένων RTT και SST-2, αντίστοιχα, εμπίπτουν σε αυτό το εύρος. Σημαντικά, το σύνολο δεδομένων SST-2 αποδίδει πιο ισχυρά αποτελέσματα στην αξιολόγηση της κλάδευσης.

Πληροφορίες για τον Δείκτη

Αξιοποιώντας το γεγονός ότι υπάρχουν πληροφορίες στα μοντέλα μετά την προσαρμογή, αναπτύξαμε τον δείκτη μας (βλέπε Ενότητα 6.4) και στις ακόλουθες φιγούρες, απεικονίζουμε τη συμπεριφορά του αλγορίθμου μας σε διάφορες επαναλήψεις. Αυτές οι απεικονίσεις δείχνουν τις βαθμολογίες προσοχής παράλληλα με τις αντίστοιχες συσχετίσεις κατά την εξέλιξη του αλγορίθμου, υπογραμμίζοντας τα κριτήρια επιλογής που καθοδηγούν τη διαδικασία κλάδευσης. Σημαντικό είναι να σημειωθεί ότι οι παρουσιαζόμενες βαθμολογίες αφορούν το κλαδευμένο μοντέλο, το οποίο αντικατοπτρίζει έναν μειούμενο αριθμό κεφαλών καθώς προχωρούν οι επαναλήψεις.

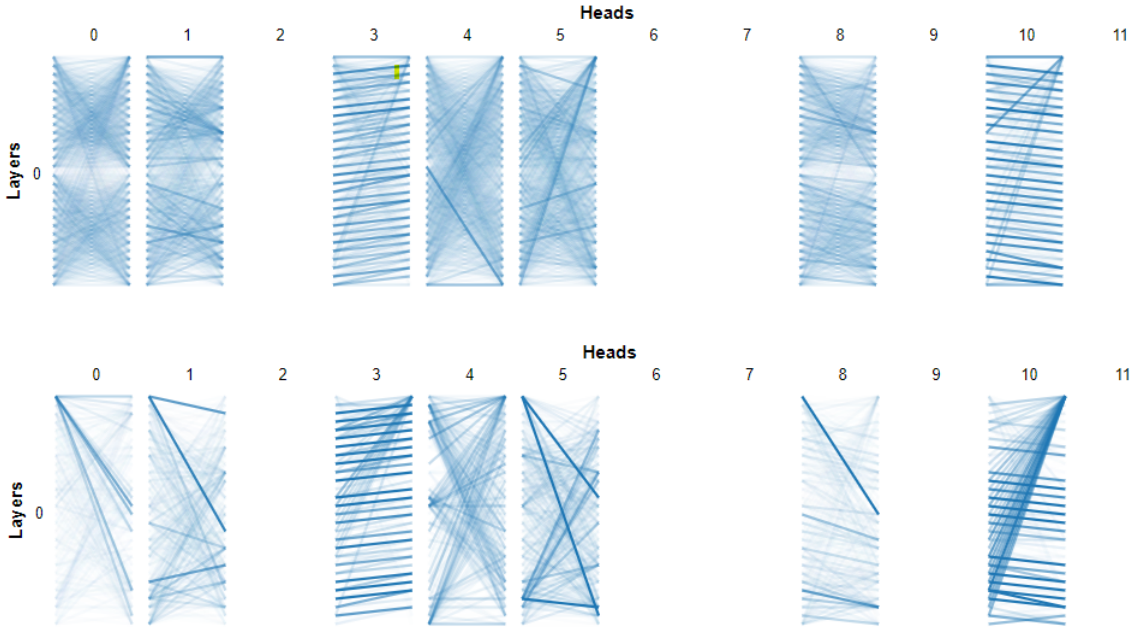


Figure 2. *Τιμές Προσοχής και Απόδοσης στην Επανάληψη 2*

Οι απεικονίσεις εστιάζουν στο Επίπεδο 1 (δείκτης 0), καθώς έχουμε ήδη βρει και εφαρμόσει τις μάσκες και προσαρμόσει το μοντέλο. Η απεικόνιση είναι μόνο για ένα δείγμα από το σύνολο δεδομένων MNLI, επομένως είναι ένα ζεύγος προτάσεων. Καθώς προχωράμε από την επανάληψη 2 στην επανάληψη 3, τα Σχήματα [6.2, 6.3], παρατηρείται μια σαφής αλλαγή στη δομή των συνδέσεων μεταξύ των βαθμολογιών προσοχής και των αντίστοιχων αποδόσεων,

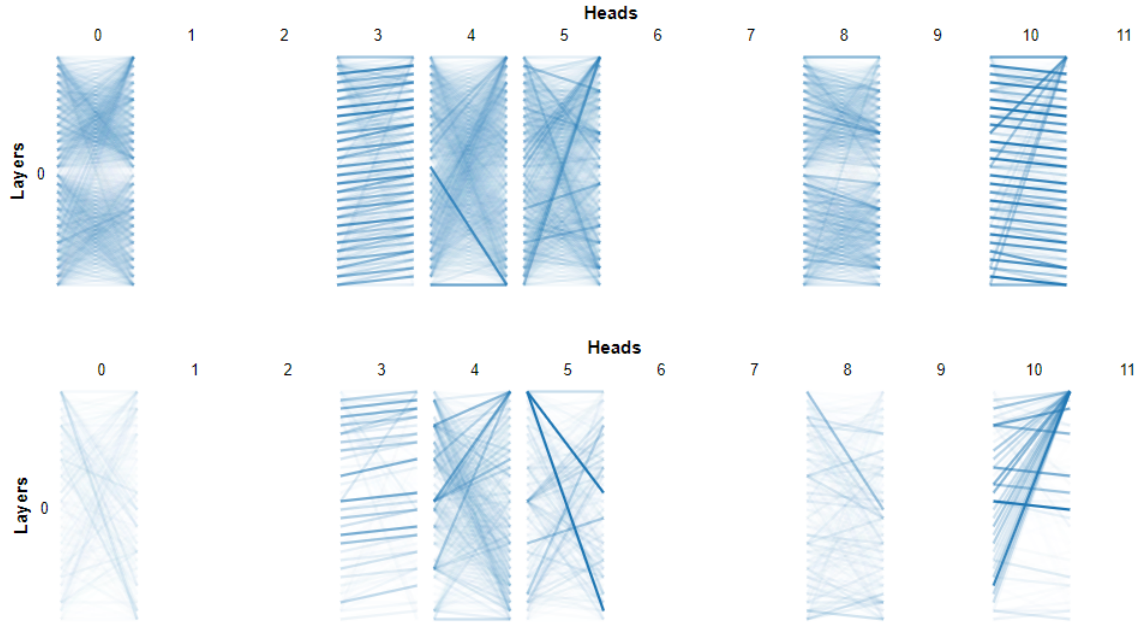


Figure 3. *Τιμές Προσοχής και Απόδοσης στην Επανάληψη 3*

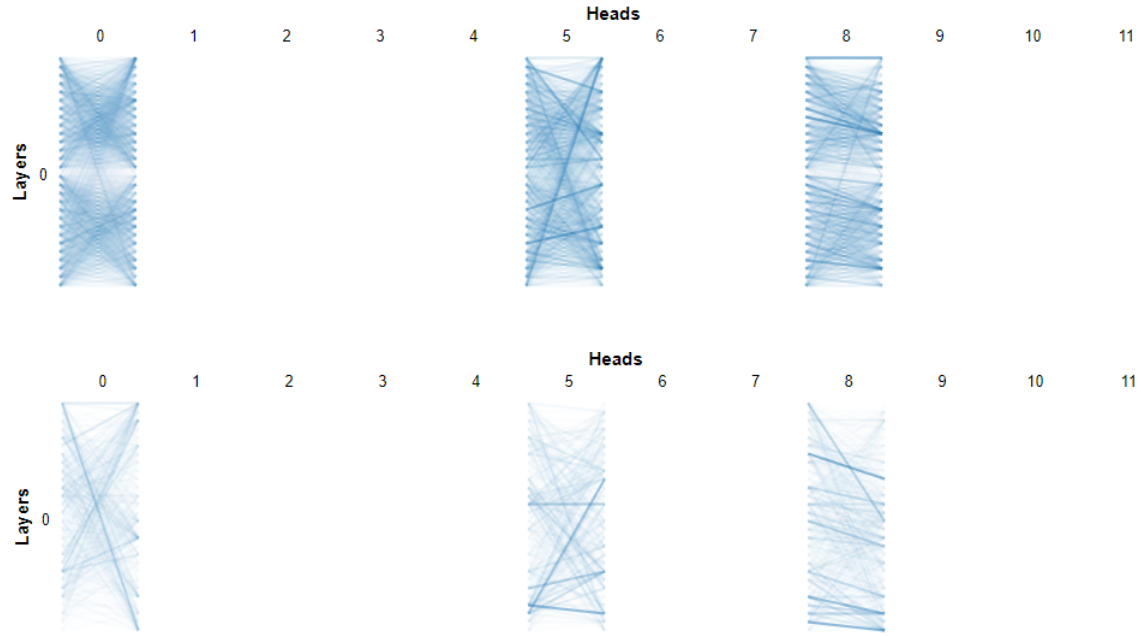


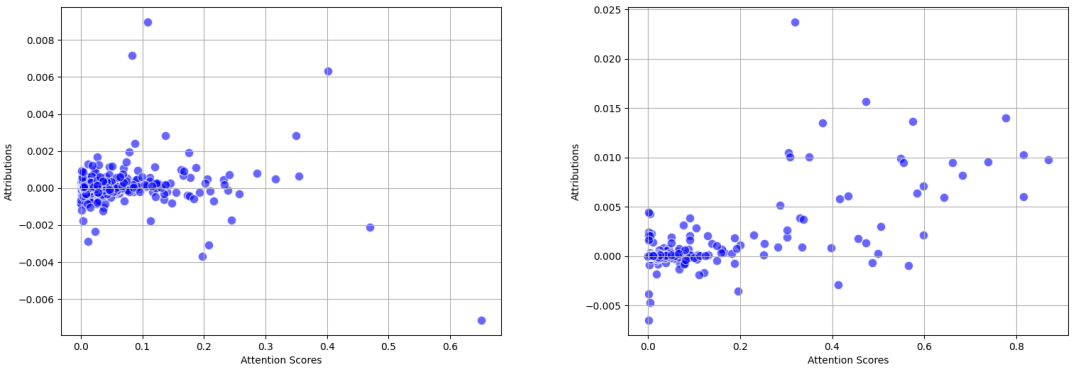
Figure 4. *Τιμές Προσοχής και Απόδοσης στην Επανάληψη 6*

ιδιαίτερα στην κεφαλή προσοχής 1. Εν τω μεταξύ, οι άλλες κεφαλές διατηρούν γενικά σταθερή δομή.

Η κύρια ανησυχία μας έγκειται στις συνδέσεις μεταξύ των τοκενών, αντί απλά στο μέγεθος των βαθμολογιών προσοχής και των αποδόσεων. Αυτή η προσέγγιση διασφαλίζει ότι η συνολική δομική ακεραιότητα μεταξύ των δύο στοιχείων διατηρείται.

Τέλος, μέχρι την 6η επανάληψη, το Σχήμα 6.4, παρατηρούμε ότι το κλαδευμένο μον-

τέλο έχει φτάσει σε σημείο σύγκλισης, όπου οι δομικές ομοιότητες μεταξύ των βαθμολογιών προσοχής και των αποδόσεων είναι σχεδόν πανομοιότυπες στις υπόλοιπες κεφαλές. Αυτή η σύγκλιση υποδηλώνει την αποτελεσματικότητα του αλγορίθμου στην επιλογή και διατήρηση κεφαλών προσοχής που συμβάλλουν ουσιαστικά στην ερμηνευσιμότητα και απόδοση του μοντέλου.



(a) Κεφαλή Προσοχής 1 *(b) Κεφαλή Προσοχής 3*

Figure 5. *Διαγράμματα Διασποράς που δείχνουν τη χωρική σχέση των κεφαλών 1, 3 στην επανάληψη 2*

Το Σχήμα 6.5 προσφέρει βαθύτερες γνώσεις σχετικά με τον δείκτη συσχέτισης κατά τη διάρκεια της επανάληψης 2. Η κεφαλή προσοχής 1 είναι ένας προφανής υποψήφιος για κλάδευση, ενώ η κεφαλή 3 φαίνεται να αξίζει να διατηρηθεί. Αυτό το συμπέρασμα είναι εμφανές από την παρατηρούμενη κατανομή των σημείων δεδομένων στο διάγραμμα διασποράς.

0.6 Πειράματα

0.6.1 Διαμόρφωση

Για τα πειράματά μας, χρησιμοποιήσαμε τα καθήκοντα ταξινόμησης από το Benchmark GLUE και συγκρίναμε τα αποτελέσματα με προηγούμενες εργασίες.

- MNLI: Multi-Genre Natural Language Inference Corpus [33]
- QQP: Σύνολο δεδομένων ερωτήσεων από το Quora
- QNLI: Ερώτηση-απάντηση NLI βασισμένο στο Stanford Question Answering Dataset [34]
- MRPC: Microsoft Research Paraphrase Corpus [35]
- SST-2: Stanford Sentiment Treebank [36]
- CoLA: Corpus of Linguistic Acceptability [37]
- RTE: Αναγνώριση Κειμενικής Συνεπαγωγής

- WNLI: Πρόκληση Σχημάτων Φυσικής Επανάληψης Winograd [38]

Έχουμε διεξάγει πειράματα στο προεκπαιδευμένο μοντέλο BERT, τόσο "bert-base-uncased" 12-στρώματα, 768-κρυμμένα, 12-κεφαλές, 110M παραμέτρους, (Βιβλιοθήκη Transformers) για διάφορες διαμορφώσεις.

Πρώτα, χρησιμοποιήσαμε ένα ισορροπημένο υποσύνολο του σετ εκπαίδευσης για κάθε εργασία για να υπολογίσουμε τον δείκτη. Στην αρχή, εκτελούμε μια στατιστική ανάλυση στα σύνολα δεδομένων για να βρούμε την περιοχή της κατανομής των δεδομένων όπου υπάρχει η περισσότερη πληροφορία, και στη συνέχεια επιλέγουμε τυχαία 2000 δείγματα από αυτά για να υπολογίσουμε τις αποδόσεις και να υπολογίσουμε τον δείκτη. Χρησιμοποιήσαμε τη μέθοδο STD ή Quantile για το φιλτράρισμα. (βλ. Παράρτημα .2 για περισσότερες λεπτομέρειες)

Η αξιολόγηση πραγματοποιείται σε ολόκληρο το σετ επικύρωσης για κάθε εργασία. Οι κλίσεις έχουν υπολογιστεί σε σχέση με την τελική πρόβλεψη, δηλαδή αφού το μοντέλο έχει ήδη λάβει την τελική απόφαση.

Διεξήγαμε διάφορα πειράματα χρησιμοποιώντας τον δείκτη κλάδευσης μας. Πρώτον, παρουσιάζουμε στις ακόλουθες φιγούρες τη συμπεριφορά κάθε αλγορίθμου και στη συνέχεια παρουσιάζουμε τα αποτελέσματά μας πιο συνοπτικά, συγκρίνοντάς τα με το έργο του Achlatis και του Michel.

Το προεκπαιδευμένο μοντέλο είναι προσαρμοσμένο με τις παραμέτρους στον Πίνακα 6.1, ίδιες με αυτές που χρησιμοποίησε ο Achlatis στο έργο του [18].

Σύνολο Δεδομένων	MNLI	QQP	QNLI	MRPC	SST-2	CoLA	RTE	WNLI
Παραδείγματα Εκπαίδευσης	392,704	363,872	104,768	3,680	67,360	8,576	2,490	635
Επαναλήψεις/Εποχή	12,272	11,371	3,274	115	2,105	268	78	20
Εποχές	3	3	3	3	3	3	3	3
Μέγεθος Παρτίδας	32	32	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}	2×10^{-5}	2×10^{-5}
Βελτιστοποιητής	AdamW με $\epsilon = 1 \times 10^{-8}$							
Μετρική Αξιολόγησης	Matched Acc.	Ακρίβεια	Ακρίβεια	Ακρίβεια	Ακρίβεια	Matthew's	Ακρίβεια	Ακρίβεια

Table 1. *GLUE tasks [15], μεγέθη συνόλων δεδομένων, μετρικές και υπερπαράμετροι προσαρμογής που αναφέρονται σε αυτή τη μελέτη.*

Εκτός από τους αλγορίθμους που πρότεινα πρόσφατα, έχω επίσης διεξάγει πειράματα σε μερικούς άλλους για σύγκριση.

1. **Μια Σημασία**: Υπολογίζουμε τον δείκτη σημασίας μόνο μία φορά και στη συνέχεια εφαρμόζουμε διάφορους λόγους κλάδευσης και αξιολογούμε στο σετ επικύρωσης. Αυτός ο αλγόριθμος χρησιμοποιήθηκε στο άρθρο του Michel [17].

2. **One Shot ISP**: Διεξάγουμε πειράματα για τον ISP για μία επανάληψη. Επομένως, υπολογίζουμε τον πρώτο δείκτη σημασίας και στη συνέχεια εφαρμόζουμε τον λόγο κλάδευσης στο προεκπαιδευμένο μοντέλο και στη συνέχεια το προσαρμόζουμε. Κάποιος θα μπορούσε να πει ότι αυτό είναι η επέκταση του αλγορίθμου "Μια Σημασία".

0.6.2 Δομημένη Κλάδευση με Δείκτη Σημασίας

Στις παρακάτω Φιγούρες 6.6, παρουσιάζουμε τα αποτελέσματά μας για όλους τους αλγορίθμους, βάζοντας τα μαζί στο ίδιο διάγραμμα.

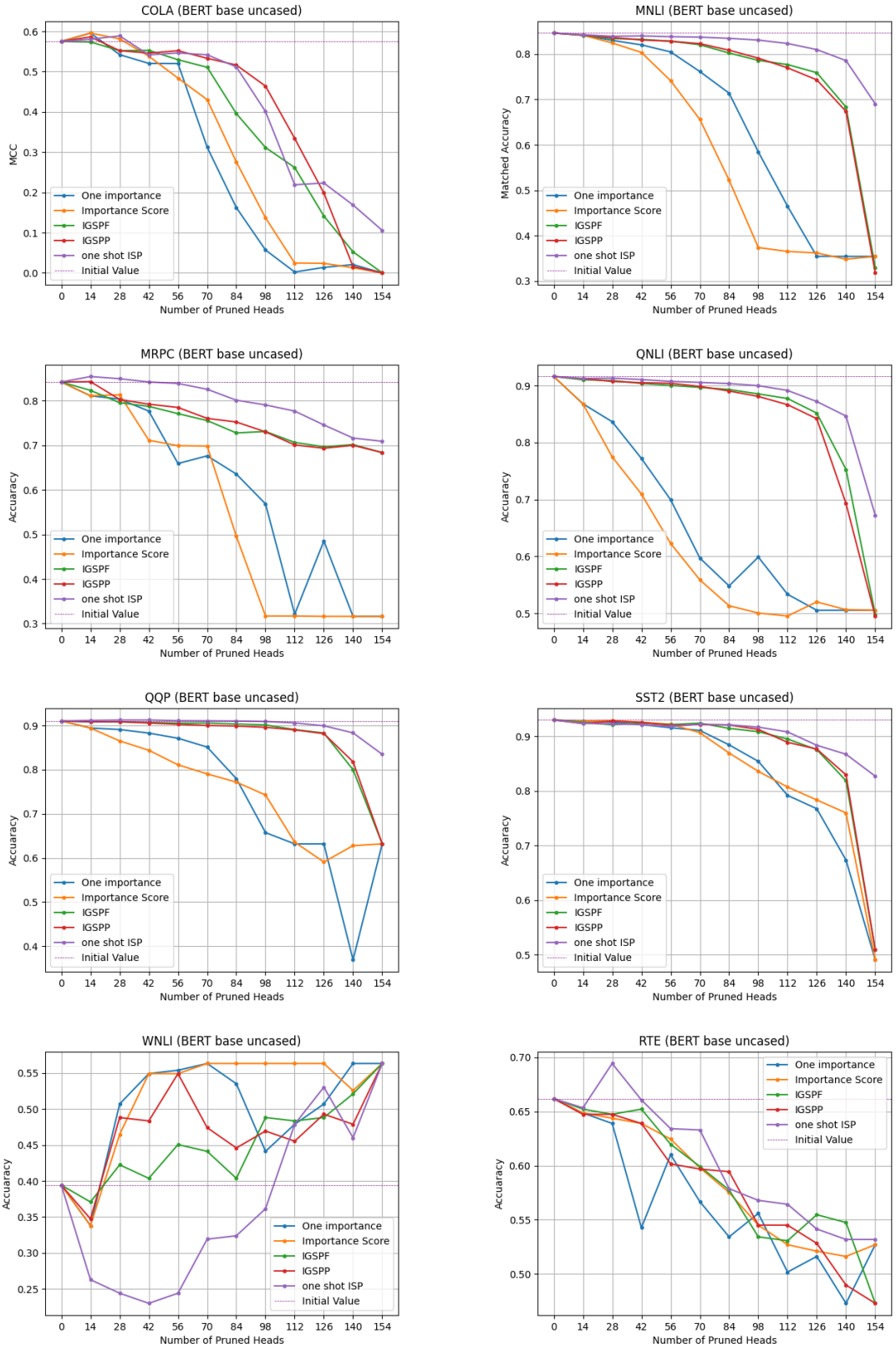


Figure 6. *Αξιολόγηση μοντέλων στο σετ επικύρωσης για κάθε εργασία για όλες τις παραλλαγές του πρώτου αλγορίθμου 6.1. Τα διαγράμματα είναι ο μέσος όρος τριών τυχαίων σπόρων.*

Όπως προτείνουν τα αποτελέσματα, οι IGSPF και IGSPP έχουν κυριολεκτικά καλύτερη απόδοση από το βασικό μοντέλο. Ωστόσο, οι 2 βασικοί αλγόριθμοι που χρησιμοποιούμε αποδίδουν αρκετά καλά σε χαμηλά επίπεδα κλάδευσης, εάν κάποιος θεωρήσει τη μείωση του χρόνου για να βρει τα υποδίκτυα, καθώς δεν χρειάζονται προσαρμογή.

Γενικά, σε μεγάλα σύνολα δεδομένων, μπορεί να παρατηρηθεί ότι ο δείκτης μας λειτουργεί αρκετά καλά, καθώς διατηρεί την απόδοση σε υψηλά επίπεδα, ακόμη και σε ποσοστά κλάδευσης 80%. Αν και οι δύο αλγόριθμοι έχουν ουσιαστικά παρόμοια συμπεριφορά, παρατηρούμε ότι ο IGSPP αποδίδει καλύτερα για μικρότερα σύνολα δεδομένων, όπως το MRPC, RTE, WNLI, CoLA.

Σημαντικό είναι η συμπεριφορά του WNLI, όπου ενώ η απόδοση μειώνεται σε όλα τα σύνολα δεδομένων, κατά τις επαναλήψεις, το WNLI βελτιώνεται σημαντικά. Μια πιθανή ερμηνεία είναι ότι το BERT είναι υπερπαραμετροποιημένο για ένα τόσο μικρό σύνολο δεδομένων, το οποίο έχει αρνητικό αποτέλεσμα. Μειώνοντας τον αριθμό των κεφαλών φαίνεται ότι παραμένουν μόνο οι πραγματικά χρήσιμες. Αυτό είναι ιδιαίτερα εμφανές στους βασικούς αλγορίθμους, όπου κόβουν τις κεφαλές κατά την εκτέλεση, το οποίο δείχνει ότι οι πολλές κεφαλές κατά την εκτέλεση περισσότερο διασπούν το μοντέλο παρά το βοηθούν να λάβει απόφαση. Επίσης, διαπιστώνουμε ότι ο IGSPP δίνει καλύτερα αποτελέσματα, το οποίο είναι λογικό, αφού προσαρμόζει μόνο τις χρήσιμες κεφαλές που προκύπτουν από τον δείκτη από την αρχή.

Αυτές οι διαφορές στις αποδόσεις μεταξύ των συνόλων δεδομένων δείχνουν τη διαφορική ποιότητα των δύο αλγορίθμων IGSPF, IGSPP. Σε μεγάλα σύνολα δεδομένων δεν διαφέρουν πολύ, και κάποιος θα μπορούσε να υποθέσει ότι ο όγκος των δεδομένων, όπου τα βάρη συνδυάζονται στην τελική διαμόρφωση, ευθύνεται. Έτσι, τα βάρη που θεωρούνται σημαντικά στο τέλος, είτε τα βρίσκεις μέσω της προσαρμογής είτε μέσω της κλάδευσης στο Προεκπαιδευμένο και στη συνέχεια της προσαρμογής, θα καταλήξουν να είναι οι ίδιες ομάδες. Σε μικρά μη-ποιοτικά σύνολα δεδομένων, όπου τα βάρη δεν έχουν συγκλίνει, ο IGSPP αποδίδει σχετικά καλύτερα καθώς αποκλείει συνεχώς τα βάρη από το αρχικό μοντέλο και ξεκινά τη διαδικασία αλλάζοντας την κατανόηση των βαρών που θα εκπαιδευτούν.

Ωστόσο, προσθέσαμε επίσης το *one shot ISP* (δείτε τον Αλγόριθμο 6.2) στο ίδιο σχήμα για να δούμε τη βελτίωση. Είναι προφανές ότι υπερβαίνει τους άλλους και διατηρεί υψηλή απόδοση ακόμα και σε βαθιά ποσοστά κλάδευσης, όπως 90%.

0.6.3 Επαναληπτική Δομημένη Κλάδευση

Μετά από αυτό, διεξάγαμε πειράματα για τον Αλγόριθμο ISP (δείτε τον Αλγόριθμο 6.2). Τα πειράματα που έγιναν αφορούν μόνο τα σύνολα δεδομένων MNLI, QNLI, καθώς απαιτούν μεγάλους υπολογιστικούς πόρους και χρόνο και δοκιμάστηκαν μόνο για ένα σπόρο.

Παρατηρήσαμε ότι η διαδικασία ISP σε αυτή τη διαμόρφωση δεν υπερβαίνει την εκδοχή one-shot. Αυτό υποδηλώνει ότι ο δείκτης σημασίας που αποκτήθηκε στην πρώτη επανάληψη είναι εξαιρετικά αποτελεσματικός, καθώς φαίνεται να καταγράφει σημαντική ποσότητα κρίσιμων πληροφοριών. Είναι σημαντικό να τονιστεί ότι η εκδοχή one-shot του ISP είναι ουσιαστικά ο αλγόριθμος "Μια Σημασία" εφαρμοσμένος στο πλαίσιο της Υπόθεσης του Λαχείου.

Αυτό το εύρημα είναι ιδιαίτερα πολύτιμο, καθώς υποδηλώνει ότι ίσως δεν χρειάζεται να εκτελούμε εκτεταμένα, απαιτητικά πειράματα για να βελτιστοποιήσουμε το μοντέλο, ενδε-

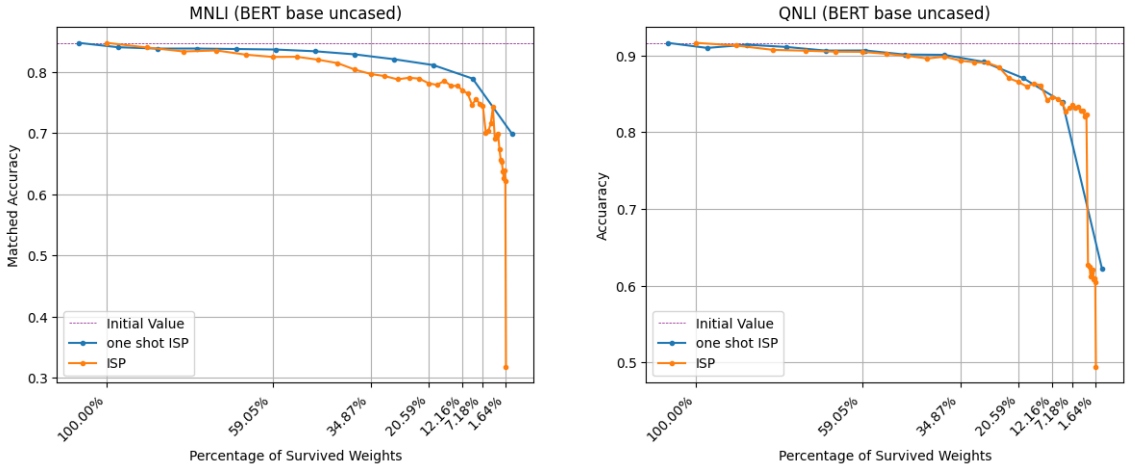


Figure 7. *Αξιολόγηση μοντέλων στο σετ επικύρωσης για τις παραλλαγές ISP (δείτε τον Αλγόριθμο 6.2). Τα πειράματα διεξήχθησαν για έναν σπόρο.*

χομένως εξοικονομώντας σημαντικό χρόνο και υπολογιστικούς πόρους.

0.6.4 Νικητήρια Εισιτήρια;

Για ορισμένους Αλγόριθμους, μας ενδιαφέρει αν μπορούμε να βρούμε τα νικητήρια εισιτήρια αξιοποιώντας τους. Έτσι, πρώτα μπορεί να είναι χρήσιμο να ορίσουμε, βάσει του Chen et al. [13], τι είναι (i) ένα αντίστοιχο υποδίκτυο, (ii) νικητήριο εισιτήριο και (iii) καθολικό υποδίκτυο.

Όπως αναφέρει στο άρθρο του:

Ας είναι $A_T^t(f(x;\theta_i,\gamma_i))$ ένας αλγόριθμος εκπαίδευσης (π.χ. AdamW με υπερπαραμέτρους) για μια εργασία T (π.χ. CoLA) που εκπαιδεύει ένα δίκτυο $f(x;\theta_i,\gamma_i)$ για την εργασία T για t βήματα, δημιουργώντας το δίκτυο $f(x;\theta_{i+t},\gamma_{i+t})$. Ας είναι θ_0 τα προεκπαιδευμένα βάρη BERT. Ας είναι $\epsilon_T(f(x;\theta))$ η μετρική αξιολόγησης του μοντέλου f για την εργασία T.

Αντίστοιχο υποδίκτυο. Ένα υποδίκτυο $f(x;m\odot\theta,\gamma)$ είναι αντίστοιχο για έναν αλγόριθμο A_T^t αν η εκπαίδευση του $f(x;m\odot\theta,\gamma)$ με τον αλγόριθμο A_T^t αποδίδει μια μετρική αξιολόγησης για την εργασία T που δεν είναι χαμηλότερη από την εκπαίδευση του $f(x;\theta_0,\gamma)$ με τον αλγόριθμο A_T^t. Με άλλα λόγια:

$$\epsilon_T\left(A_T^t(f(x;m\odot\theta,\gamma))\right)\geq\epsilon_T\left(A_T^t(f(x;\theta_0,\gamma))\right)$$

Νικητήριο εισιτήριο. Ένα υποδίκτυο $f(x;m\odot\theta,\gamma)$ είναι ένα νικητήριο εισιτήριο για έναν αλγόριθμο A_T^t αν είναι αντίστοιχο υποδίκτυο για τον A_T^t και $\theta=\theta_0$.

Καθολικό υποδίκτυο. Ένα υποδίκτυο $f(x;m\odot\theta,\gamma_{T_i})$ είναι καθολικό για τις εργασίες $\{T_i\}_{i=1}^N$ αν είναι αντίστοιχο για κάθε $A_{T_i}^{t_i}$ για κατάλληλες, ειδικές ρυθμίσεις εργασίας γ_{T_i}.

Βασισμένος στον Chen, λόγω των διακυμάνσεων, για να θεωρηθεί ένα υποδίκτυο ως νικητήριο εισιτήριο, χρειάζεται η απόδοση του πλήρους μοντέλου BERT να είναι εντός μιας τυπικής απόκλισης από την απόδοση του υποδικτύου (για την ισότητα). Επομένως, η συνθήκη που χρησιμοποιούμε είναι ότι η απόδοση του πλήρους μοντέλου πρέπει να είναι μικρότερη από το ανώτατο όριο της απόδοσης του υποδικτύου.

Σύνολο Δεδομένων	QNLI	QQP	MRPC	WNLI	RTE	SST-2	CoLA
Αραιότητα IGSPF				100%	30%		10%
Αραιότητα IGSPP			10%	100%	30%	30%	10%
Αραιότητα one shot ISP	20%	60%	30%	100%	30%		20%
Πλήρες $\text{BERT}_{\text{BASE}}$	91.5 ± 0.06	91.0 ± 0.07	84.2 ± 1.2	39.4 ± 0.8	66.1 ± 1.2	93.0 ± 0.3	57.5 ± 0.9
$f(x, m_{\textbf{IGSPF}} \odot \theta_0)$				56.3 ± 0.0	65.2 ± 2.2		57.3 ± 1.4
$f(x, m_{\textbf{IGSPP}} \odot \theta_0)$			84.3 ± 1.1	56.3 ± 0.0	63.8 ± 3.3	92.5 ± 0.4	58.6 ± 0.3
$f(x, m_{\textbf{one shot ISP}} \odot \theta_0)$	91.2 ± 0.2	91.0 ± 0.05	84.2 ± 0.3	56.3 ± 0.0	66.0 ± 1.6		58.8 ± 0.9

Table 2. *Νικητήρια Εισιτήρια σε όλη την έκταση των συνόλων δεδομένων για τους IGSPF, IGSPP και one shot ISP*

Μπορεί να παρατηρηθεί ότι η προσέγγιση one-shot ISP (Επαναληπτική Αραιωτική Κλάδευση) μπορεί να εντοπίσει νικητήρια εισιτήρια σε υψηλά επίπεδα αραιότητας σε σύνολα δεδομένων όπου οι άλλοι αλγόριθμοι αποτυγχάνουν να τα εντοπίσουν εντελώς.

Μετά από αυτό, συγκρίνουμε τα αποτελέσματά μας με τα καλύτερα αποτελέσματα του Achlatis, ως σημείο αναφοράς. Τα αποτελέσματα δεν έχουν αναπαραχθεί, επομένως η σύγκριση είναι κατά προσέγγιση και ποιοτική.

Σε αυτό το σημείο, πρέπει να τονίσουμε ότι ο υπολογισμός της σημασίας στο έργο του Achlatis χρησιμοποιεί το σετ επικύρωσης, ενώ εμείς χρησιμοποιούμε ένα ισορροπημένο υποσύνολο του σετ εκπαίδευσης.

Για να ευθυγραμμιστούμε με τα πειράματα του Achlatis, στον Πίνακα 6.3 παρουσιάζουμε τα αποτελέσματα για ποσοστό κλάδευσης 70% και για την απλή έκδοση χωρίς την εφαρμογή LTH, ενώ στον Πίνακα 6.4 υπάρχουν τα αποτελέσματα της LTH για ποσοστό κλάδευσης 80%.

Εργασία	MNLI	QNLI	QQP	SST-2	MRPC	CoLA
Michel($\alpha = 1.0$)	0.717	0.734	0.791	0.875	0.639	0.286
Achlatis	**0.732**(0.4)	**0.787** (0.7)	**0.811** (0.6)	**0.878** (0.4)	**0.730** (0.6)	**0.387** (0.5)
Δικό μας	0.374	0.500	0.742	0.836	0.317	0.137

Table 3. *Μετρικές απόδοσης για διαφορετικές εργασίες υπό μεθόδους κλάδευσης και αξιολόγησης του προσαρμοσμένου μοντέλου. Ο μεγαλύτερος αριθμός είναι με έντονη γραφή.*

Εργασία	MNLI	QNLI	MRPC	SST-2	CoLA
Michel ($\alpha = 1.0$)	0.817	0.820	0.773	0.903	0.329
Achlatis	**0.824** (0.6)	**0.880** (0.4)	**0.783** (0.4)	**0.918** (0.4)	**0.435** (0.5)
(Δικό μας) IGSPF	0.776	<u>0.877</u>	0.706	0.895	0.261
(Δικό μας) IGSPP	0.769	<u>0.866</u>	0.701	0.889	<u>0.334</u>

Table 4. *Μετρικές απόδοσης για διαφορετικές εργασίες υπό μεθόδους, εφαρμόζοντας την Υπόθεση του Λαχείου. Ο μεγαλύτερος αριθμός είναι με έντονη γραφή, ενώ υπογραμμίζουμε τις τιμές που υπερβαίνουν την προσέγγιση του Michel.*

Ο δείκτης μας στον απλό αλγόριθμο δεν μπόρεσε να υπερβεί την προηγούμενη εργασία και είναι σημαντικά λιγότερο αποδοτικός. Ωστόσο, εφαρμόζοντας την LTH, τα αποτελέσματα είναι καλύτερα, ξεπερνώντας την προσέγγιση του Michel για ορισμένες εργασίες. Στον Πίνακα 6.4, προσθέσαμε επίσης την τροποποιημένη εκδοχή του Αλγορίθμου 6.1, χρησιμοποιώντας το προεκπαιδευμένο μοντέλο.

Chapter 1

Introduction

1.1 Introductory Concepts

1.1.1 Artificial Intelligence, Machine Learning and Deep Learning

Artificial Intelligence (AI) refers to the simulation of human intelligence in machines designed to think and act like humans. AI systems can perform tasks such as problem-solving, decision-making, understanding natural language, and recognizing patterns. These systems can be either rule-based, following predefined instructions, or learning-based, where they improve their performance by learning from data [39].

Machine Learning (ML) is a subset of AI that involves training algorithms on data to enable them to make predictions or decisions without being explicitly programmed. ML techniques allow computers to learn from and adapt to new data. The main types of ML are supervised learning, where the model learns from labeled data; unsupervised learning, where the model identifies patterns in unlabeled data; and reinforcement learning, where the model learns by receiving rewards or penalties based on its actions [40].

Deep Learning (DL) is a specialized branch of ML that uses neural networks with many layers, hence "deep." These networks can automatically discover representations needed for feature detection or classification from raw data. DL is particularly powerful for tasks such as image and speech recognition and natural language processing. For instance, Convolutional Neural Networks (CNNs) are widely used for image-related tasks, while Transformer models like BERT are used for text-based applications [1].

1.1.2 Interpretability in Machine Learning

Interpretability in ML refers to the degree to which a human can understand the cause of a decision made by a model. As ML models, especially those involving DL, become more complex, interpretability becomes essential for several reasons:

1. *Debugging and Improving Models*: Understanding model decisions can help identify errors and improve model performance.

2. *Building Trust*: Users are more likely to trust and accept ML models if they understand how decisions are made.

3. *Ensuring Fairness*: Interpretability helps detect and mitigate biases in ML models, promoting ethical AI use.

4. *Regulatory Compliance*: In many sectors, regulations require transparency and explainability for automated decisions [41].

Interpretability techniques include feature importance measures, model visualization, and methods like Integrated Gradients, which attribute the contribution of each input feature to the model's output [22].

1.1.3 Pruning in Machine Learning

Pruning is a model compression technique used to reduce the size and complexity of ML models by eliminating less important components, such as neurons or layers. The primary goals of pruning include:

1. *Reducing Model Size*: Pruning decreases the number of parameters, making models smaller and more manageable.

2. *Improving Efficiency*: Smaller models require less computational power and memory, enhancing execution speed and making them suitable for deployment on resource-constrained devices.

3. *Enhancing Generalization*: Pruning can help prevent overfitting by removing unnecessary components, leading to better performance on unseen data [42].

Pruning can be categorized into:

1. *Unstructured Pruning*: Removes individual parameters, resulting in sparse matrices.

2. *Structured Pruning*: Removes entire structures like neurons or layers, leading to more efficient models with minimal performance loss.

3. *Enhancing Generalization*: Pruning can help prevent overfitting by removing unnecessary components, leading to better performance on unseen data [42].

Combining pruning with interpretability ensures that the most crucial parts of the model are retained, maintaining performance while achieving significant efficiency gains. This approach is especially beneficial for deploying sophisticated models like BERT in environments with limited resources.

1.2 Motivation

Transformers have revolutionized the field of Natural Language Processing (NLP), achieving remarkable performance across a variety of tasks. Despite their success, training these models is computationally intensive, and they often contain millions of parameters. The base BERT model, for instance, has 110 million parameters, but more recent models like GPT-4 boast even more parameters (OpenAI, 2023). The development continues

with models like Google's Gemini and Anthropic's Claude, which further push the limits of size and capability. This rapid growth has sparked concerns about computational complexity, environmental impact, and fairness in comparing different architectures, as well as reproducibility [43][44][45].

GPT-4, developed by OpenAI, follows in the footsteps of its predecessor GPT-3, with even more parameters and capabilities. GPT-4 excels in understanding and generating human-like text, producing more coherent and contextually accurate responses [19]. This model sets new benchmarks in various NLP tasks such as text completion, translation, summarization, and question answering, highlighting significant advancements in language model technology.

Google's Gemini represents another significant leap in language model development. It incorporates the latest advancements in AI research, including architectural improvements and advanced training techniques, to deliver state-of-the-art performance across diverse NLP applications. Gemini is designed to be highly efficient, maintaining high accuracy while addressing the computational and environmental concerns associated with large language models [20].

Anthropic's Claude is a state-of-the-art language model that emphasizes safety and interpretability. Claude aims to provide robust and reliable NLP capabilities while ensuring that its decisions are understandable and trustworthy. This focus on interpretability is crucial for applications where transparency and accountability are essential [21].

Current Challenges and Solutions

Human language is incredibly complex, and current models might not be using their parameters as efficiently as possible. Research by Voita et al. (2019) [11] demonstrated that most Transformer heads could be pruned without significant performance loss. Similarly, Clark et al. (2019) [46]found that many heads within the same BERT layer show similar self-attention patterns, which explains why Michel et al. (2019) [17]could reduce most layers to a single head without degrading performance. Prasanna et al. (2020) [12] extended this line of inquiry in their paper "When BERT Plays the Lottery, All Tickets Are Winning," where they showed that different subsets of the model parameters could be pruned while still maintaining performance. This further emphasizes the overparameterization of these models and suggests that there is significant redundancy within BERT's structure, reinforcing the potential for effective model compression through pruning.

For certain tasks, some BERT heads and layers are not only redundant but can also be detrimental. Disabling specific heads has shown positive effects in machine translation [17], abstractive summarization [47], and various GLUE tasks [8]. Tenney et al. (2019) [48] found that some layers could decrease performance in structured probing tasks, particularly the final layers. Unstructured pruning research by Gordon et al. (2020) [49] found that 30-40% of the weights could be pruned without impacting downstream tasks.

While larger models generally perform better, this isn't always the case. For instance, BERT-base has outperformed BERT-large on tasks like subject-verb agreement [50] and sentence subject detection [51]. The complexity of language and the extensive pre-training data contribute to redundancy in BERT's heads and layers. [46] suggest that attention dropouts, which zero out some attention weights during training, might contribute to this

redundancy.

Given the evidence of overparameterization, it's not surprising that models like BERT can be efficiently compressed with minimal accuracy loss, which is highly desirable for real-world applications. Various techniques, such as Knowledge Distillation [52], Pruning [53] , and Neural Network Quantization [54], can achieve this compression. This study focuses on Pruning, specifically Structured Pruning.

We explore Structured Pruning over Magnitude Pruning because adaptive sparse matrix multiplication has shown promising results on GPUs but has not yet been widely applied in silicon. Structured pruning remains the most effective method for accelerating models like BERT in practice. Accelerating unstructured sparse matrix multiplication is an ongoing research area, with recent progress achieving near-ideal speed-ups with minimal deviation from unstructured sparsity [55].

Structured pruning also allows us to investigate the roles of different components within the neural network. This understanding can improve fine-tuning techniques and enhance the overall functionality and efficiency of the network.

1.2.1 Research Contribution

In this study we examine a structured pruning approach for BERT-based architectures focusing on the attributions of the attention weights of the fine-tuned model. Furthermore, we study this method through Lottery Ticket Hypothesis, where we see that concluding in competitive results. As Achlatis mentioned in his Thesis, we also apply "Iterative Structured Pruning".

1.3 Thesis Outline

Chapter 2: Machine 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: Interpretabiliy, discusses about various techniques that make these complex models more understandable to humans. This includes methods for visualizing and explaining model decisions, understanding feature importance, and evaluating the transparency of different architectures. We also discuss the significance of interpretability in ensuring fairness, building trust, and complying with regulatory requirements.
Chapter 6: Attribution Does Matter: 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 attention weights and the corresponding attributions, being calculated through Integrated Gradients technique, achieving good performance. 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.
Chapter 7: Conclusions & Future Work, contains our conclusion, summarizing our findings and providing an outlook into the future work.

Part I

Background Knowledge

Chapter 2

Machine Learning

2.1 Introduction

In this chapter, we introduce the theoretical background of Machine Learning (ML). Chapters 2.1 and 2.2 present the definition and categories of Machine Learning, respectively. Chapter 2.3 provides an introduction to traditional Machine Learning models, and Chapter 2.4 explores Neural Networks and Deep Learning. A recommended book for an introduction to Deep Learning is "Deep Learning" by Goodfellow, Bengio, and Courville (2016) [1].

2.1.1 Definition

Machine Learning (ML) is a subfield of Artificial Intelligence (AI) and was first defined in 1959 by Arthur Samuel as "the field of study that gives computers the ability to learn without being explicitly programmed." A formal definition was later provided in 1997 by Tom Mitchell, who proposed that "a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance on tasks in T, as measured by P, improves with experience E" (Mitchell, 1997) [56]. Therefore, Machine Learning focuses on the design and implementation of algorithms and systems that automatically train and optimize. The training of a model is performed on a set of input data, and using statistical analysis, the model is optimized to more efficiently recognize patterns in observations or make better decisions

2.2 Machine Learning Classifications

Before classifying the types of learning in machine learning, a proper definition of "learning" should be provided. Indeed, Mitchell [56] 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

Supervised learning is a machine learning task that involves learning a function that maps an input to an output based on example input-output pairs. In this context, the training data consists of input feature vectors along with their corresponding target (ground truth) vectors, which are well-labeled, meaning that each input is tagged with the correct output. During training, the algorithm learns the mapping function from the input variables X to the output variable Y (expressed as $Y = f(X)$), using both X and their corresponding y provided in the training data. At inference, this learned function allows the algorithm to predict the output for new, unseen inputs that come from the same distribution as the training samples. Supervised learning problems can be categorized into two main types: classification and regression. In classification, the target variable is a discrete class label, such as identifying handwritten digits from 0 to 9 in an image. In regression, the target is one or more continuous variables, such as predicting the value of a function $y = f(x)$ for given input features x. Essentially, supervised learning involves a supervisor or teacher that provides the correct answers during training, enabling the model to learn from this labeled data and make accurate predictions on new data. (Bishop, 2006) [57].

2.2.2 Unsupervised Learning

Unsupervised learning is a category of machine learning problems where the training data consists of input vectors x without any corresponding target values. Unlike supervised learning, where the goal is to learn a mapping from inputs to known outputs, unsupervised learning aims to uncover the underlying structure or distribution in the data to learn more about it. This type of learning is termed "unsupervised" because there are no correct answers provided during training, and no teacher to guide the learning process.

A significant subclass of unsupervised learning tasks involves clustering, where the objective is to group observations so that members of the same group are similar to each other and different from those in other groups. This process relies on defining and measuring similarities between feature vectors and selecting an appropriate clustering method to group the samples based on these similarities (Bishop, 2006) [57].

Another important class of unsupervised tasks is generative modeling. Generative models attempt to imitate the process that generates the training data, with the aim of producing new data that resembles the training data. This is considered unsupervised learning because the data generation process is not directly observable – only the data itself is.

Additionally, unsupervised learning can involve tasks like density estimation, where the goal is to determine the distribution of data within the input space, or dimensionality reduction, where high-dimensional data is projected into lower dimensions for visualization or simplification. Association problems, another subset of unsupervised learning, seek to discover rules that describe large portions of the data without the existence of labels, further illustrating the diverse applications of unsupervised learning in modeling and understanding complex datasets.

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 [58].

2.2.4 Transfer Learning

The ideal scenario in machine learning is to have abundant labeled training instances that share the same distribution as the test data. However, obtaining sufficient labeled data is often expensive, time-consuming, or even unrealistic in many scenarios. Semi-supervised learning addresses this issue by utilizing a limited amount of labeled data along with a large

amount of unlabeled data to improve learning accuracy. Despite this, collecting unlabeled data can also be challenging, resulting in traditional models being unsatisfactory.

Transfer learning emerges as a promising solution to these challenges by focusing on transferring knowledge across domains. The concept of transfer learning is inspired by educational psychology, particularly the generalization theory of transfer proposed by psychologist C.H. Judd. According to this theory, learning to transfer is the result of the generalization of experience. For instance, a person who has learned to play the violin can learn the piano faster than someone with no musical training, because both instruments share some common knowledge. Similarly, transfer learning leverages knowledge from a source domain to enhance learning performance or reduce the number of labeled examples needed in a target domain.

However, it's important to note that the effectiveness of transferred knowledge is not guaranteed. If there is little commonality between the source and target domains, the transfer may not be beneficial and could even be detrimental. This underscores the need for careful consideration when selecting source and target domains for transfer learning.

Key Concepts in Transfer Learning

Domain: A domain D consists of two components: a feature space X and a marginal distribution $P(X)$. Thus, $D = \{X, P(X)\}$. The feature space X is an instance set defined as $X = \{x \mid x_i \in X, i = 1, \ldots, n\}$.

Task: A task T includes a label space Y and a decision function f, represented as $T = \{Y, f\}$. The decision function f is an implicit function expected to be learned from sample data.

Transfer Learning: Given some observations from $m_S \in \mathbb{N}^+$ source domains and tasks $\{(D_{S_i}, T_{S_i}) \mid i = 1, \ldots, m_S\}$ and observations from $m_T \in \mathbb{N}^+$ target domains and tasks $\{(D_{T_j}, T_{T_j}) \mid j = 1, \ldots, m_T\}$, transfer learning utilizes the knowledge from the source domain(s) to improve the performance of the decision functions f_{T_j} on the target domain(s). When $m_S = 1$, this is referred to as single-source transfer learning, and when $m_S > 1$, it is called multi-source transfer learning. In most scenarios, $m_T = 1$.

Transfer learning leverages knowledge from one or more source domains to address the limitations posed by insufficient labeled data in the target domain, making it a vital approach in modern machine learning [59].

2.3 Learning Process

2.3.1 Loss Function

We will now review some fundamental loss functions widely utilized in machine learning. Loss functions map an input or some variables to a real number, which signifies the cost to be minimized.

Regression Loss Functions

Regression problems involve training a model to fit a curve $f(x)$ to given points. The dataset consists of pairs of points (x_i, y_i), where $i = 1, \ldots, N$. The model must learn to map an input point x_i to its corresponding value y_i using an appropriate function $f(x)$. For each point x_i, the true value is y_i and the predicted value from the model is $f(x_i)$.

- **Squared Error Loss**: Squared Error Loss is the squared difference between the actual and predicted values:

$$L = (y - f(x))^2 \tag{2.1}$$

- **Absolute Error Loss**: Absolute Error Loss is the absolute value of the difference between the actual and predicted values:

$$L = |y - f(x)| \tag{2.2}$$

Classification Loss Functions

In classification tasks, each input feature vector has an associated label. The model takes the training sample as input and predicts its label. The primary loss function used in these problems is Cross Entropy Loss. Entropy quantifies the uncertainty involved with certain probability distributions; more uncertainty/variation in a probability distribution leads to higher entropy. The entropy of a (discrete) distribution p is defined as:

$$H(p) = -\sum_x p(x) \log(p(x)) \tag{2.3}$$

The negative sign ensures the quantity is positive, since $p(x) \leq 1 \Rightarrow \log(p(x)) < 0$. Cross Entropy between two distributions p and q measures their difference. In terms of information theory, it can be viewed as the distance between the two distributions, based on the amount of information (bits) needed to describe that distance. It is defined as:

$$H(p, q) = -E_p[\log(q)] \tag{2.4}$$

For discrete distributions, this relation is equivalent to:

$$H(p, q) = -\sum_x p(x) \log(q(x)) \tag{2.5}$$

In machine learning, we assume the true probability of each pattern i is p_i, and q_i is the model's predicted value for this pattern. For instance, in a binary classification problem where each pattern belongs to one of two classes (0 or 1), the model's output can be interpreted as the probability that the current pattern, with input vector x, belongs to class 1. This probability is modeled by the logistic function:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

where z is the actual output of the model, i.e., a function of the input vector x ($z = f(x)$). Therefore, the probability that the label of x is 1 is:

$$q_{y=1} = \hat{y} = \frac{1}{1 + e^{-f(x)}} \tag{2.7}$$

and thus, the probability that its label is 0 is:

$$q_{y=0} = 1 - \hat{y} \tag{2.8}$$

This implies that $p \in \{y, 1-y\}$ (where y is the actual label, $y \in \{0, 1\}$) and $q \in \{y, 1-\hat{y}\}$. The cross entropy of these two distributions can now be used to estimate their difference:

$$H(p, q) = -\sum_i p_i \log(q_i) = -y \cdot \log(\hat{y}) - (1 - y) \cdot \log(1 - \hat{y}) \tag{2.9}$$

The larger the difference between y and $\hat{y}$, the higher the value of the cross entropy loss function. For example, if $y = 1$ and $\hat{y} \approx 0$, then the first term of equation (8) will be large. Conversely, if $\hat{y} \approx 1$, the first term is almost zero. This cross entropy loss function is used in binary classification problems but can be generalized for multi-class classification problems as follows:

$$H(p, q) = -\sum_i p_i \log(q_i) = -\sum_i y_i \log(\hat{y}_i) \tag{2.10}$$

Additionally, y does not need to take only discrete values. If not a classification task, y can take any value between 0 and 1, similar to $\hat{y}$. Hence, cross entropy loss is also used to measure reconstruction error in cases such as autoencoders (see Section 2.5.4.2).

Another useful loss function that measures the difference between two distributions p and q is Kullback-Leibler divergence:

$$KL(p||q) = \sum_x p(x) \log\left(\frac{p(x)}{q(x)}\right) \tag{2.11}$$

If p and q are similar, their ratio is close to 1, making the KL divergence near zero, indicating small differences. Conversely, if they are quite dissimilar, the KL divergence will be larger. Interestingly, we have:

$$KL(p||q) = H(p, q) - H(p) \tag{2.12}$$

Since p is the target distribution, $H(p)$ remains the same regardless of q, allowing us to omit this term and just calculate $H(p, q)$ (cross entropy loss). This is why cross entropy loss is more commonly used than KL divergence.

2.3.2 Optimization

Optimization involves selecting the most suitable model parameters to minimize the loss function, which is scalar. Minimizing a loss function $L(a)$ with respect to a parameter vector a can be achieved using a gradient descent procedure. This iterative method updates

the parameter vector by taking small steps in the direction of the negative gradient of the loss function.

We start with an arbitrary initial parameter vector $a^{(1)}$ and compute the gradient vector $\nabla L(a^{(1)})$. The next value $a^{(2)}$ is obtained by moving some distance from $a^{(1)}$ in the direction of steepest descent. Generally, $a^{(k+1)}$ is calculated from $a^{(k)}$ using the equation:

$$a^{(k+1)} = a^{(k)} - \eta^{(k)} \nabla L(a^{(k)}) \tag{2.13}$$

Here, η is a positive scale factor called the learning rate, which determines the step size and, consequently, how "steep" the actual update is. We aim for this sequence of parameter vectors to converge to a solution that minimizes $L(a)$. Thus, the number of updates (iterations) required is problem-dependent. A common stop criterion is that the update quantity $\eta^{(k)} \nabla L(a^{(1)})$ is smaller than a threshold θ, chosen by the user.

One of the most critical aspects of gradient descent procedures is selecting the learning rate $\eta^{(k)}$. If $\eta^{(k)}$ is too small, convergence will be slow; if too large, the update can overshoot and even diverge.

Another interesting aspect of gradient descent methods in machine learning is that we choose the loss function L based on the training set, requiring the entire dataset for each step to compute ∇L. Techniques that use the entire dataset at each iteration are called batch methods. An online version of gradient descent, called stochastic gradient descent, is useful for large datasets. This method updates the parameter vector based on one data point at a time, for example, by examining each point sequentially. The most common approach, however, is to use a batch of data points to compute ∇L, combining the advantages of both methods: addressing large datasets while incorporating parallelism through groups of samples at each step, speeding up the training process [60, 57].

Gradient descent optimization algorithms are numerous (Adaline, Adamax, Adam, RMSprop, etc.), each introducing slight variations to the classical approach. One widely used algorithm is Adam (Adaptive Moment Estimation), which updates the weights using estimations of the first two moments of past gradients (mean and standard deviation).

2.3.3 Gradient Descent

Gradient descent is a method to minimize a loss function $J(\theta)$ parameterized by a model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ with respect to the parameters. The learning rate λ determines the size of the steps the algorithm takes to reach a (local) minimum. In other words, the model follows the slope of the surface created by the objective function downhill until it reaches a valley, as depicted in Figure 2.1.

There are three variants of gradient descent, which differ in the amount of data used to compute the gradient of the loss function. Depending on the amount of data, there is a trade-off between the accuracy of the update and the time it takes to perform it.

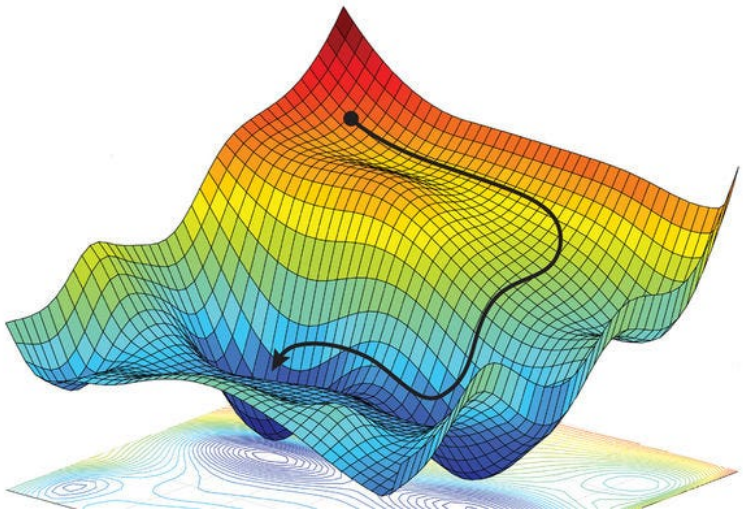


Figure 2.1. *Gradient Descent Visualization.*

Batch Gradient Descent

Batch gradient descent computes the gradient of the cost function with respect to the parameters θ for the entire training dataset:

$$\theta = \theta - \lambda \cdot \nabla_\theta J(\theta) \tag{2.14}$$

While the gradient descent algorithm is straightforward to implement, it is not guaranteed to converge to a global minimum. It is possible to get stuck at a local minimum if the learning rate is not properly chosen. Moreover, when training on large datasets, computing the loss over the entire dataset at each iteration may be computationally expensive and inefficient.

Stochastic Gradient Descent (SGD)

Stochastic gradient descent is a variant of gradient descent. Unlike gradient descent, it computes the gradient of the loss function over a subset of samples rather than the whole dataset. Thus, SGD estimates the gradient using a sample, instead of computing the true gradient using all samples. For each training example x_i and its corresponding label y_i, a parameter update is performed as:

$$\theta = \theta - \lambda \cdot \nabla_\theta J(\theta; x_i, y_i) \tag{2.15}$$

While batch gradient descent converges to the minimum of the basin that the parameters are placed in, SGD enables it to jump to new and potentially better local minima. On the other hand, this complicates convergence to the exact minimum, as SGD may keep overshooting. However, it has been shown that when the learning rate is slowly decreased, SGD exhibits the same convergence behavior as batch gradient descent, almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively [61].

Mini-batch Gradient Descent

Mini-batch gradient descent takes the best of both worlds and performs an update for every mini-batch of n training examples:

$$\theta = \theta - \lambda \cdot \nabla_\theta J(\theta; x_{i:i+n}, y_{i:i+n}) \tag{2.16}$$

where the mini-batch is denoted as $x_{i:i+n}, y_{i:i+n}$. This approach reduces the variance of the parameter updates, leading to more stable convergence. It also leverages highly optimized matrix operations in deep learning libraries, making the computation of the gradient with respect to a mini-batch very efficient.

Learning Rate

The size of the steps the optimization algorithm takes plays a critical role in training the model and effectively controls the speed at which the model learns. If the step or learning rate is too large, we may overshoot the minima and bounce back and forth on the loss surface, or the model can become unstable and diverge. If it is too small, it may take too long to reach a minimum, or worse, the algorithm might get stuck at a suboptimal local minimum. Finding an optimal learning rate is challenging, and in practice, it often comes down to experimenting with the model's hyperparameters.

Specifying a learning rate for each epoch, i.e., scheduling, can mitigate these issues to some extent. There are two types of methods for scheduling global learning rates: decay and cyclical methods. The preferred method is learning rate annealing, which gradually decreases the learning rate during the training process. A relatively large step-size is preferred at the initial stages of training to achieve better generalization [62]. The cyclical method [63] involves repeating a learning rate period that consists of an upper and lower bound during epochs. The observation behind the cyclic method is that while increasing the learning rate in the optimization process may have a negative effect, it can result in better generalization of the trained model.

Learning rate warmup is a recent approach that uses a relatively small step size at the beginning of training. The learning rate is then increased linearly or non-linearly to a specific value in the first few epochs and then gradually shrinks to zero. The observations behind warmup are that the model parameters are initialized using a random distribution, and thus, the initial model is far from ideal; therefore, a large learning rate causes the model to recede from a local minimum. Also, carefully training an initial model in the first few epochs may enable the application of a larger learning rate in the middle stage of training, resulting in better regularization.

2.3.4 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 on which 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 minimized. The generalization error is defined as the expected value of the error on new inputs. 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 include its ability to minimize the training error and reduce the gap between training and test errors. 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. Therefore, in machine learning, we aim to find a good trade-off between the training error and the gap between training and test errors, as illustrated in Figure 2.2.

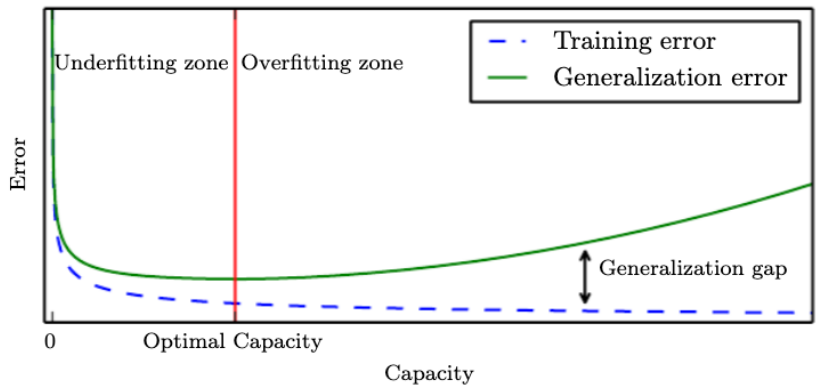


Figure 2.2. *Training and test errors behave differently. At the left end of the graph, training error and generalization error are both high, indicating underfitting. As we increase model capacity, training error decreases, but the gap between training and generalization error increases. Eventually, the size of this gap outweighs the decrease in training error, and we enter the overfitting regime, where capacity is too large. Source: [1]*

2.3.5 Regularization, Dropout, and Pruning

To overcome the problem of overfitting and improve generalization, we employ techniques such as regularization, dropout, and pruning. Our modern ideas about improving the generalization of machine learning models are refinements of thoughts dating back to philosophers at least as early as Ptolemy. Many early scholars invoked a principle of parsimony, 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 to machine learning problems. To address the potential loss of generalization, we impose restrictions on the form of the solution by forcing the model to choose the simplest solution in terms of parameters. This is done by adding a term to the loss equation that penalizes the size of the model. Thus, the loss function takes the following form:

$$\hat{\theta} = \arg\min_{\theta} L(\theta) = \arg\min_{\theta} \left(\frac{1}{N} \sum_{i=1}^{N} L(f(x_i; \theta), y_i) + \lambda R(\theta) \right) \tag{2.17}$$

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. Regularization inherently intends to 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 a 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 L2 and L1.

L2 Regularization

L2 regularization involves the standard Euclidean norm (L2-norm) of the parameters, aiming to keep the sum of the squares of the parameter values low. Large model weights $W[i, j]$ will be penalized since they are considered "unlikely". L2 regularization is often referred to as weight decay. High weights are severely penalized, but weights with small values are only negligibly affected.

$$R_{L2}(W) = \|W\|_2^2 = \sum_{i,j} W[i, j]^2 \tag{2.18}$$

L1 Regularization

The L1 regularizer punishes uniformly low and high values and intends to decrease all non-zero parameter values towards zero. It encourages sparse solutions, meaning models with many parameters set to zero. The L1 regularizer is also called a sparse prior or lasso.

$$R_{L1}(W) = \|W\|_1 = \sum_{i,j} |W[i, j]| \tag{2.19}$$

Dropout

An effective technique for preventing neural networks from overfitting the training samples is dropout training. Dropout is designed to prevent the network from relying on specific weights. It randomly sets to zero (drops) half of the neurons in the network (or in a specific layer) for each training example during stochastic gradient training. The dropout technique is one of the key factors contributing to the robust results of neural networks.

Pruning

Pruning is another method to prevent overfitting. It involves removing connections between neurons or entire neurons, channels, or filters from a trained network by zeroing out values in its weights matrix or removing groups of weights entirely. We will discuss pruning in more detail in Chapter 4.

2.4 Machine Learning Models

2.4.1 Linear Regression

Linear Regression is a Supervised Learning technique aimed at solving regression problems. The goal is to construct a model that takes a vector $\mathbf{x} \in \mathbb{R}^n$ as input and predicts a scalar $y \in \mathbb{R}$ as its output. The output of linear regression is a linear combination of the input features.

Let $\hat{y}$ be the predicted value of y. We define the prediction as:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} \tag{2.20}$$

where $\mathbf{w} \in \mathbb{R}^n$ is a vector of coefficients. These coefficients are parameters that govern the behavior of the model. In this context, w_i represents the weight assigned to feature x_i before summing up the contributions from all the features. We can consider $\mathbf{w}$ as a set of weights that determine the influence of each feature on the prediction. To optimally fit the model to the training data, we aim to minimize the Mean Squared Error (MSE) on the training set:

$$\text{MSE}_{\text{train}} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 \tag{2.21}$$

To find the minimum $\text{MSE}_{\text{train}}$, we solve for where its gradient is zero:

$$\nabla_{\mathbf{w}} \text{MSE}_{\text{train}} = 0 \implies \mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \tag{2.22}$$

Thus, for Linear Regression, we have a closed-form solution to determine the optimal parameters. However, it is often more practical to compute $\mathbf{w}$ using gradient descent because the closed-form solution can be computationally intensive in terms of time and memory. Additionally, gradient descent allows for parallelization. Furthermore, gradient descent is preferred when the data is not linearly separable because the matrix $(\mathbf{X}^\top \mathbf{X})^{-1}$ may not be invertible in such cases.

2.4.2 Classifiers

As mentioned earlier, classifiers are utilized in supervised learning applications with the aim of assigning a sample x to a category y. This means that for each sample x_i, where $i = 1, \ldots, N$, the goal is to identify the class y_k, where $k = 1, \ldots, M$, that maximizes the probability $p(y_k|x_i)$ according to decision theory:

$$\hat{y} = \arg\max_{y_k} p(y_k|x_i) \tag{2.23}$$

There are two types of classifiers that attempt to calculate this probability: discriminative models and generative models.

- **Discriminative models** directly estimate the posterior class probabilities $p(y_k|x_i)$ and then apply decision theory (2.23) to assign each x_i to the most likely class. The most well-known discriminative models include Logistic Regression 2.4.2, Support Vector Machines (SVMs)2.4.2, perceptrons (see Section 2.5.1), and traditional neural networks (see Section 2.5.2).

- **Generative models** approach the calculation of posterior class probabilities differently. They first determine the class-conditional probability densities $p(x_i|y_k)$ for each class y_k and then the prior probabilities $p(y_k)$. Using Bayes' rule, the posterior probability is calculated as follows:

$$p(y_k|x_i) = \frac{p(x_i|y_k)p(y_k)}{p(x_i)} = \frac{p(x_i|y_k)p(y_k)}{\sum_k p(x_i|y_k)p(y_k)} \tag{2.24}$$

  Alternatively, generative models can calculate the joint probability $p(x_i, y_k)$ and then obtain the posterior probability by normalizing. Common generative models include Naive Bayes classifiers, Bayesian Networks, and Hidden Markov Models (HMMs).

It is evident that generative models not only learn a decision boundary, as discriminative models do, but also the underlying distribution of each class. The choice between the two categories depends on the specific task and application, though discriminative models are generally more popular. Logistic Regression and SVMs, in particular, are widely used.

Logistic Regression is a linear model which, in a binary classification problem, assumes $p(y_1|x)$ can be expressed as a logistic sigmoid function applied to a linear combination of the feature vector x:

$$p(y_1|x) = \sigma(\mathbf{w}^\top \mathbf{x}) \tag{2.25}$$

This linear function $\mathbf{w}^\top \mathbf{x}$ serves as the decision boundary (discriminant function) of the classifier, meaning that points on one side of this hyperplane belong to class y_1 and those on the other side to class y_2. Naturally, $p(y_2|x) = 1 - p(y_1|x)$. The parameters of the model, the components of vector $\mathbf{w}$, are computed using Maximum Likelihood Estimation and the Cross Entropy Loss function (see Section 2.3.1).

Support Vector Machines (SVMs) are also linear models that seek to determine the weight vector of the linear discriminant function $g(x) = \mathbf{w}^\top \mathbf{x}$. The objective of SVMs is to find the weights that maximize the margin between the hyperplane defined by the linear function and the training samples of the two classes, which is given by:

$$\frac{g(x)}{\|\mathbf{w}\|} \tag{2.26}$$

Generally, the larger the margin, the better the generalization capability of the classifier. SVMs can also address the problem of non-linearly separable data by transforming the samples into a space where they become linearly separable through a transformation ϕ. In this case, $z = \phi(x)$ is the transformed vector, and the linear discriminant function becomes $g(z) = \mathbf{w}^\top z$ [57, 60].

Logistic Regression

Logistic Regression (LR) is a simple yet effective classification algorithm that is widely used for binary classification tasks. The primary goal is to predict the probability that a given input vector $\mathbf{x} \in \mathbb{R}^n$ belongs to a particular class, resulting in a probability value between 0 and 1.

Consider a binary classification scenario with input vectors $\mathbf{x}_i$ and corresponding labels $y_i \in \{0, 1\}$. Logistic Regression models the probability of the positive class using the logistic sigmoid function applied to a linear combination of the input features:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.27}$$

For a given input vector $\mathbf{x}$, the model's prediction is defined as:

$$h_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}} \tag{2.28}$$

To fit the model to the training data, we minimize the following loss function, which combines the regularization term with the logistic loss:

$$\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \log\left(1 + \exp\left(-y_i(\mathbf{w}^\top \mathbf{x}_i + b)\right)\right) \tag{2.29}$$

Here, C is a regularization parameter that controls the trade-off between fitting the training data and keeping the model parameters small, and b is the bias term. Logistic Regression is computationally efficient and provides a good baseline for many Natural Language Processing (NLP) tasks. However, it is limited by its linear decision boundary, making it unsuitable for problems where the classes are not linearly separable. A common extension for multi-class classification involves using the One-vs-One (OvO) approach, where binary classifiers are trained for each pair of classes.

Support Vector Machines (SVMs)

In many machine learning applications, the feature vectors of different classes are not linearly separable in their original space. This means it can be challenging to find a hyperplane in the input feature space that serves as a classification boundary for the data belonging to each class in the training set. To overcome this, the original finite-dimensional space can be mapped into a higher-dimensional space, making separation easier in that space [64].

SVMs seek to find maximum-margin hyperplanes to create these classification boundaries between the vectors of each class, as depicted in Figure 2.3. Let the training set

consist of N input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$ with corresponding target values $y_1, \ldots, y_N$ where $y_i \in \{-1, 1\}$.

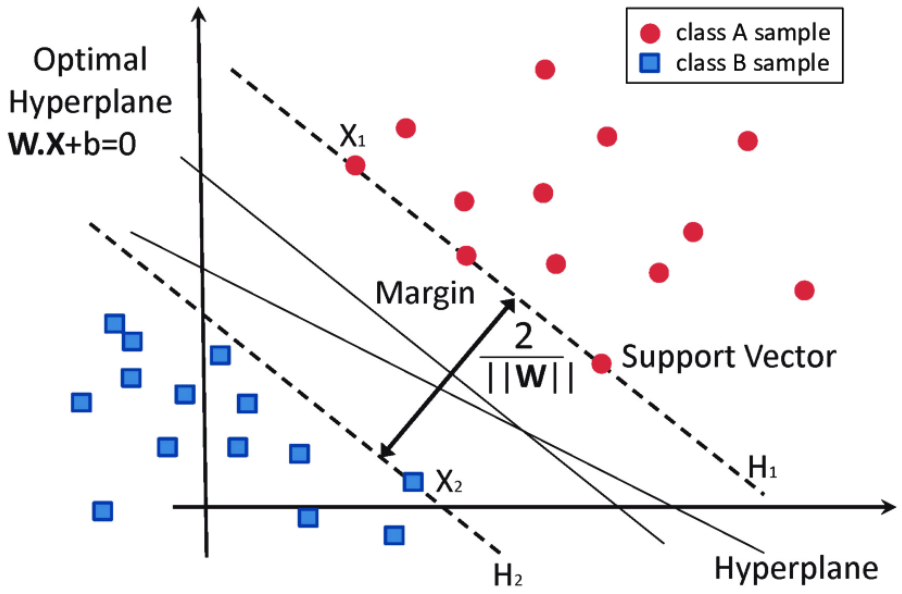


Figure 2.3. *Example of binary classification using SVM, showing the maximum-margin hyperplane and support vectors.*

We are given l training examples $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, l$, where each example $\mathbf{x}_i \in \mathbb{R}^d$, and a class label with one of two values $y_i \in \{-1, 1\}$. All hyperplanes in $\mathbb{R}^d$ are parameterized by a vector $\mathbf{w}$ and a constant b, expressed by the equation:

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.30}$$

Given such a hyperplane $(\mathbf{w}, b)$ that separates the data, this defines the function:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.31}$$

which correctly classifies the training data and other unseen data. The canonical hyperplane is defined as one that separates the data from the hyperplane by a "distance" of at least 1. Therefore, we consider those that satisfy:

$$y_i(\mathbf{x}_i \cdot \mathbf{w} + b) \geq 1 \quad \forall i \tag{2.32}$$

To calculate the geometric distance from the hyperplane to a data point, we normalize by the magnitude of $\mathbf{w}$. This distance is simply:

$$d((\mathbf{w}, b), \mathbf{x}_i) = \frac{y_i(\mathbf{x}_i \cdot \mathbf{w} + b)}{\|\mathbf{w}\|} \geq \frac{1}{\|\mathbf{w}\|} \tag{2.33}$$

Intuitively, we want the hyperplane that maximizes the geometric distance to the closest data points. From the equation, we see this is accomplished by minimizing $\|\mathbf{w}\|$ subject to the distance constraints. This is typically achieved using Lagrange multipliers. We define the matrix $(H)_{ij} = y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$, and introduce more compact notation. The problem

transforms into:

$$\min_{\alpha} W(\alpha) = -\alpha^T \mathbf{1} + \frac{1}{2}\alpha^T H \alpha \tag{2.34}$$

subject to:

$$\alpha^T \mathbf{y} = 0 \tag{2.35}$$

$$0 \leq \alpha \leq C\mathbf{1} \tag{2.36}$$

where α is the vector of l non-negative Lagrange multipliers to be determined, and C is a regularization term that penalizes misclassified instances. From the derivation, the optimal hyperplane can be written as:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \tag{2.37}$$

The solution to the constrained equation system (2.34, 2.35, 2.36) is given by the Lagrange multipliers [57].

When a dataset is not linearly separable, it does not mean there isn't another way to separate the data. To address this, we define a mapping $\mathbf{z} = \phi(\mathbf{x})$ that transforms the d-dimensional input vector $\mathbf{x}$ into a higher d'-dimensional vector $\mathbf{z}$. In the new optimization problem, we replace all occurrences of $\mathbf{x}$ with $\phi(\mathbf{x})$. The problem becomes:

$$\min_{\alpha} W(\alpha) = -\alpha^T \mathbf{1} + \frac{1}{2}\alpha^T H \alpha \tag{2.38}$$

with $(H)_{ij} = y_i y_j (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j))$. Equation (2.13) becomes:

$$\mathbf{w} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i) \tag{2.39}$$

Whenever $\phi(\mathbf{x}_a)$ appears, it is always in a dot product with another $\phi(\mathbf{x}_b)$. If we know the kernel function $K(\mathbf{x}_a, \mathbf{x}_b) = \phi(\mathbf{x}_a) \cdot \phi(\mathbf{x}_b)$, the matrix in our optimization problem becomes $(H)_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$. Our classifier then becomes:

$$f(\mathbf{x}) = \text{sign}\left(\sum_i \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right) \tag{2.40}$$

We can extend the binary SVMs to multi-class problems by training separate binary classifiers for each pair of classes in the training data and selecting the one with the highest confidence.

2.5 Deep Learning Models

As previously mentioned, every model or learning technique requires input feature vectors. The process of feature extraction is dependent on the specific problem, making it challenging to identify the most suitable representations that will facilitate solving the learning problem, whether it is supervised or unsupervised. One approach is to utilize machine learning not only to train the computer to map feature vectors to the desired

output but also to learn the representation itself. In this context, deep learning provides a solution for extracting high-level features from raw data by introducing representations that are defined in terms of simpler ones. This enables the computer to build complex concepts from basic ones. Deep learning, a subset of machine learning, is inspired by the structure and function of the human brain and the way humans think [1].

The core idea of deep learning is the use of multiple elemental non-linear computing units, known as artificial neurons, arranged in networks. The interconnections of these networks resemble the way neurons are interconnected in the human brain. These networks are referred to as neural networks, and they are trained through successive presentations of training patterns.

Interest in neural networks dates back to the development of learning machines called perceptrons during the 1950s and 1960s, which mimicked the way brain neurons function. Mathematical proofs demonstrated that perceptrons could converge to a solution after a finite number of steps when trained with linearly separable data. The solution involved finding the parameters (coefficients) of hyperplanes that could separate the data into classes representing the training samples.

However, perceptrons struggled with non-linearly separable data, prompting the idea of using multilayer perceptrons. These could potentially learn to separate data with more complex relationships. The true breakthrough was that these networks could now learn representations more suitable for recognizing the input data. Each layer of the network refines the representations to more abstract levels, a process known as deep learning, which is particularly effective with large datasets.

The effective training method that enables the learning of these representations is called backpropagation. While it lacks the mathematical rigor in guaranteeing convergence to a solution, as seen with single-layer perceptrons, it has yielded impressive results in pattern recognition.

Although neural networks are highly autonomous in their training, they still require human intervention for parameter tuning. Configurable parameters include the number of layers, the number of neurons per layer, and other problem-specific coefficients. Deep learning does not always provide the optimal solution; many applications are better suited to traditional methods. However, deep learning has proven extremely valuable in applications that have been challenging for other methods. It has enabled solutions to many problems across various domains, including speech recognition, natural language processing and understanding, and genetics [2].

Next, we will delve into how perceptrons function and how they are combined to create neural networks.

2.5.1 The Perceptron

Biological Neurons vs Perceptrons

The perceptron is a mathematical model designed to mimic the functionality of a biological neuron.

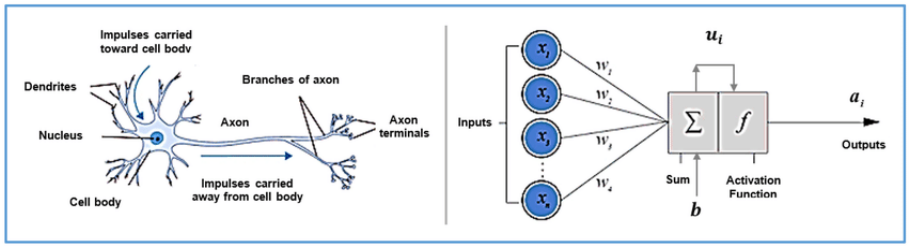


Figure 2.4. *A biological neuron compared to a perceptron*

- In biological neurons, dendrites receive electrical signals from the axons of other neurons. In perceptrons, these signals are represented as numerical values. These values x_i, $i = 1, 2, \ldots, n$, form the elements of an n-dimensional input vector $\mathbf{x}$.
- At the synapses, where dendrites connect with axons, the strength of the electrical signals is modulated. In perceptrons, this modulation is modeled by multiplying each input value x_i by a weight w_i.
- A biological neuron fires an output signal if the combined strength of the input signals exceeds a certain threshold. In perceptrons, this combined strength is represented as the weighted sum of the input values. The weighted sum can be expressed in three different forms:

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_{n+1} = \sum_{i=1}^{n} w_i x_i + w_{n+1} = \mathbf{w}^\top \mathbf{x} + w_{n+1} \tag{2.41}$$

A step function is then applied to this sum to determine whether the output will be 1 (activated) or -1 (not activated). The output f of the neuron is defined by the following equation:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^\top \mathbf{x} + w_{n+1} > 0 \\ -1 & \text{if } \mathbf{w}^\top \mathbf{x} + w_{n+1} < 0 \end{cases} \tag{2.42}$$

Perceptron in Linear Classification Problems

It is evident that the weighted sum computed in the perceptron corresponds to a linear boundary (hyperplane) in an n-dimensional space:

$$\mathbf{w}^\top \mathbf{x} + w_{n+1} = 0 \tag{2.43}$$

Here, $\mathbf{w}$ (also referred to as the weight vector) and $\mathbf{x}$ are n-dimensional column vectors, and $\mathbf{w}^\top \mathbf{x}$ is their inner product. This implies that a single perceptron unit can be employed to solve a classification problem by learning this linear boundary between linearly separable pattern classes.

If we add a 1 to the end of every pattern vector, then $\mathbf{x} = [x_1, x_2, \ldots, x_n, 1]^\top$ and $\mathbf{w} = [w_1, w_2, \ldots, w_n, w_{n+1}]^\top$. Therefore, the classification problem between two linearly

separable pattern classes c_1 and c_2 is to find a set of weights $\mathbf{w}$ such that, given an input vector $\mathbf{x}$, the following property holds:

$$\mathbf{w}^\top \mathbf{x} = \begin{cases} > 0 & \text{if } \mathbf{x} \in c_1 \\ < 0 & \text{if } \mathbf{x} \in c_2 \end{cases} \tag{2.44}$$

In this formulation, $\mathbf{x}$ and $\mathbf{w}$ are referred to as augmented pattern and weight vectors, respectively. The solution to this problem is provided by an iterative algorithm, which, according to the perceptron convergence theorem, will converge to a solution (a set of weights that define a hyperplane) after a finite number of steps, provided the pattern classes are linearly separable.

The perceptron training algorithm is straightforward. Let α denote the learning rate, which defines the steepness of weight vector updates in each iteration. The initial values of the weight vector, denoted by $\mathbf{w}^{(1)}$, are arbitrary. Assuming our dataset consists of N patterns $\mathbf{x}_j$, $j = 1, 2, \ldots, N$, we perform the following for $k = 2, 3, \ldots$:

For each pattern vector $\mathbf{x}_j$ at step k:

1. If $\mathbf{x}_j \in c_1$ and $\mathbf{w}^\top \mathbf{x}_j \leq 0$, let:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \alpha \mathbf{x}_j \tag{2.45}$$

2. If $\mathbf{x}_j \in c_2$ and $\mathbf{w}^\top \mathbf{x}_j \geq 0$, let:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha \mathbf{x}_j \tag{2.46}$$

3. Otherwise, let:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} \tag{2.47}$$

The concept behind this algorithm is that if a pattern is misclassified, we adjust the weight vector to increase the likelihood of correct classification the next time the specific pattern is presented. If the classification of a pattern is correct, no change is made to the weight vector. The algorithm converges and terminates at step K when all patterns in our dataset can be correctly classified using the current weight vector $\mathbf{w}^{(K)}$ [2].

2.5.2 Fully Connected Neural Network

To learn complex non-linear functions, architectures that combine multiple artificial neurons can be developed and implemented. These architectures are known as Multi-Layer Perceptrons (MLPs). An MLP is a type of feedforward artificial neural network (FFNN) that consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Except for the input nodes, each node is a neuron that employs a non-linear activation function. The use of multiple layers and non-linear activation functions distinguishes MLPs from linear perceptrons, enabling them to handle data that is not linearly separable. The architecture of such a network is presented in Figure 2.5. A deep neural network has more than one hidden layer.

Each neural network comprises the following layers:

- **Input layer**: This layer receives the input data and passes information from the external environment to the network without any additional computation. The nodes in this layer relay the information to the hidden layer.
- **Hidden layer(s)**: One or more hidden layers preprocess the inputs received from the previous layer. They extract essential features from the input data. As data moves through higher hidden layers, more abstract features are constructed.
- **Output layer**: After preprocessing the data, this layer makes a decision based on the processed information.

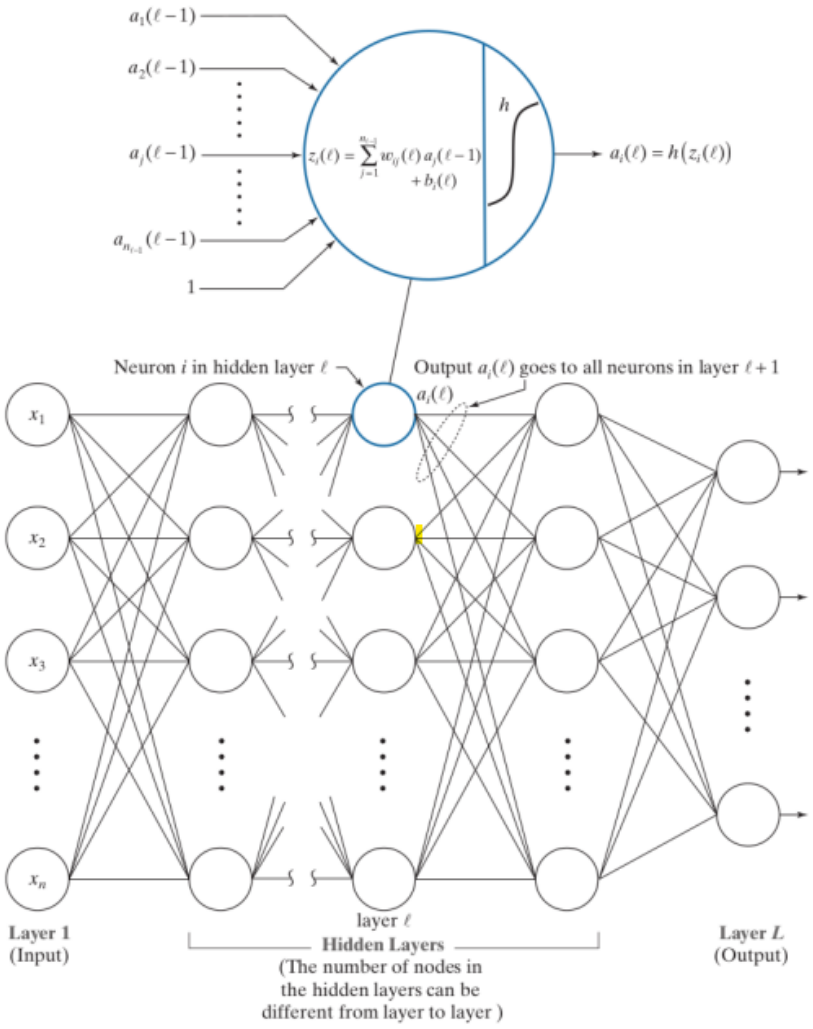


Figure 2.5. *Fully Connected Neural Network [2]*

2.5.3 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a type of artificial neural network where connections between units form a directed cycle, enabling the network to maintain an internal

state that allows for dynamic behavior. Proposed in the 1980s, RNNs can utilize their internal memory to process and handle arbitrary sequences of inputs, making them suitable for tasks such as unsegmented connected handwriting recognition, speech recognition, natural language processing, and machine translation.

RNNs are particularly effective with sequential data because each neuron can use its internal memory to retain information about previous inputs. This means that RNNs have a memory state that captures information about what has been processed so far. Figure 2.6 shows the architecture of a rolled-up recurrent neural network. An RNN can be thought of as multiple copies of the same network, each passing a message to its successor. Figure 2.7 illustrates what happens when we unroll the loop.

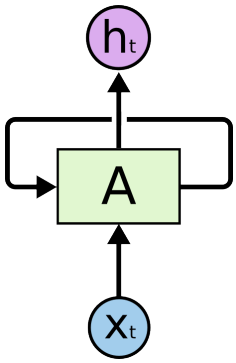


Figure 2.6. *A rolled-up RNN where X_t is the input vector containing sequences of characters of a word while h_t is the output vector. Source: colah.github.io*

Consider a natural language example: "I had washed my house" is different from "I had my house washed". This difference allows the network to gain a deeper understanding of language statements. This is important because, when reading a sentence, even humans pick up the context of each word from the words before or after it. An RNN has loops that allow information to be carried across neurons while reading the 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 in h_t. The goal is to use h_t as output and compare it to test data (which is usually a small subset of the original data). Then we obtain error rate information. After comparing the output to the test data, with the error rate in hand, we use Back Propagation Through Time (BPTT) to backpropagate through the network and adjust the weights based on the error rate, allowing the network to learn and improve its performance.

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

$$h_t = f_h(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \tag{2.48}$$

$$y_t = f_y(W_{yh}h_t + b_y) \tag{2.49}$$

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, b_y is the bias for y, and f_x, f_h are the activation functions for x and h respectively. There are three separate matrices of weights: W_{hx} (input-to-hidden weights), W_{hh} (hidden-to-hidden weights), and W_{yh} (hidden-to-output weights).

Long Short-Term Memory (LSTM): RNNs can handle context from the beginning of the sequence, allowing for more accurate predictions of a word at the end of a sequence. However, in practice, RNNs are limited to looking back only a few steps. This is why RNNs need to be used with Long Short-Term Memory (LSTM) units, introduced by Hochreiter and Schmidhuber [65]. Adding LSTM to the network introduces a memory unit that can remember context from the very beginning of the input. For example, if a sentence contains 10 words and we want to predict the 11th word, all 10 words are processed by the RNN, and their weights at each step are saved using LSTM, allowing for accurate prediction of the 11th word. Vanilla RNNs also face the issue of vanishing and exploding gradients through BPTT.

A memory cell is used in addition to the hidden layer to pass information that might not be used immediately but is crucial for prediction. These memory units enable RNNs to produce more accurate results by retaining the context across inputs. Figure 2.8 illustrates the repeating module in an LSTM.

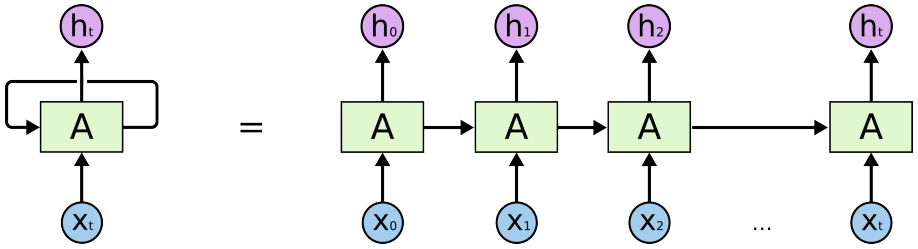


Figure 2.7. *An unrolled RNN where X_t is the input vector containing sequences of characters of a word while h_t is the output vector. Source: colah.github.io*

Given a sequence $x_1, x_2, \ldots, x_t, \ldots, x_n$ of vectors of an input sequence of length n, for vector x_t, with inputs h_{t-1} and c_{t-1}, h_t and c_t are computed as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.50}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.51}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.52}$$

$$u_t = \tanh(W_u x_t + U_u h_{t-1} + b_u) \tag{2.53}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot u_t \tag{2.54}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.55}$$

- **Forget gate (f_t)**: This gate determines which information should be discarded or

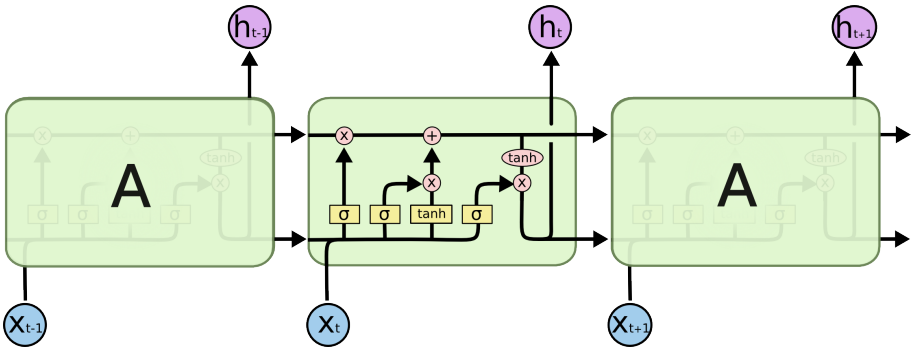


Figure 2.8. *The repeating module in an LSTM. Source: colah.github.io*

retained. Information from the previous hidden state h_{t-1} and the current input x_t is passed through a sigmoid activation function. Values close to 0 indicate forgetting, while values close to 1 indicate retaining the information.

- **Input gate (i_t)**: The previous hidden state and current input are passed into a sigmoid function to decide which values will be updated. The hidden state and current input are also passed to a tanh function to squash values between -1 and 1 (u_t). The tanh output is then multiplied by the sigmoid output ($i_t \odot u_t$) to filter the important information.

- **Cell state (c_t)**: The cell state is updated by pointwise multiplication with the forget vector, dropping values if they are near 0. The output from the input gate is then added to update the cell state with relevant new values.

- **Output gate (o_t)**: This gate determines the next hidden state, which contains information from previous inputs and is used for predictions. The previous hidden state and current input are passed into a sigmoid function, and the new cell state is passed to a tanh function. The output is the hidden state, obtained by multiplying the tanh output by the sigmoid output ($o_t \odot \tanh(c_t)$). The new cell state and hidden state are carried over to the next time step.

2.5.4 Attention Model

Attention models were initially introduced by Bahdanau et al. [3] for machine translation and have since become a fundamental concept in neural network research. The attention mechanism is widely adopted in various applications within Natural Language Processing (NLP), speech recognition, and computer vision, owing to its effectiveness in enhancing model performance.

The idea behind attention can be compared to human cognitive systems. For example, our visual system tends to focus on specific parts of an image, filtering out irrelevant details to aid perception [66]. Similarly, in many tasks involving language, speech, or vision, certain parts of the input are more critical than others. For instance, in translation and summarization tasks, only specific words in the input sequence may be crucial for predicting the next word. Likewise, in image captioning, particular regions of an image may be more

relevant for generating the next word in the caption. The attention mechanism allows models to dynamically focus on important parts of the input, improving task performance.

The concept of attention can be traced back to the regression model proposed by Nadaraya and Watson in 1964 [3]. Given training data consisting of features and their corresponding target values, the goal is to predict the target value for a new query instance. Instead of predicting the simple average of target values, Nadaraya and Watson proposed a weighted average, where weights reflect the relevance of training instances to the query. The attention function $a(x, x_i)$ assigns weights to instances, encoding their relevance to the query. This approach ensures that the estimator is consistent and simple, as the information is contained within the data, not the weights. Modern attention mechanisms generalize this concept by learning the weighting function.

The first application of the attention model in deep learning was for sequence-to-sequence tasks by Bahdanau et al. [3]. A sequence-to-sequence model typically comprises an encoder-decoder architecture, as shown in Figure 2.9. The encoder processes an input sequence of tokens $\{x_1, x_2, \ldots, x_T\}$ into fixed-length vectors $\{h_1, h_2, \ldots, h_T\}$. The decoder then generates an output sequence $\{y_1, y_2, \ldots, y_{T'}\}$ from these vectors.

Traditional encoder-decoder frameworks face two significant challenges. First, encoding all input information into a single fixed-length vector h_T can lead to information loss, especially for long sequences [3]. Second, this framework does not model the alignment between input and output sequences, which is critical for tasks like translation and summarization. In sequence-to-sequence tasks, each output token should be influenced by specific parts of the input sequence. However, the decoder lacks a mechanism to focus on relevant input tokens dynamically.

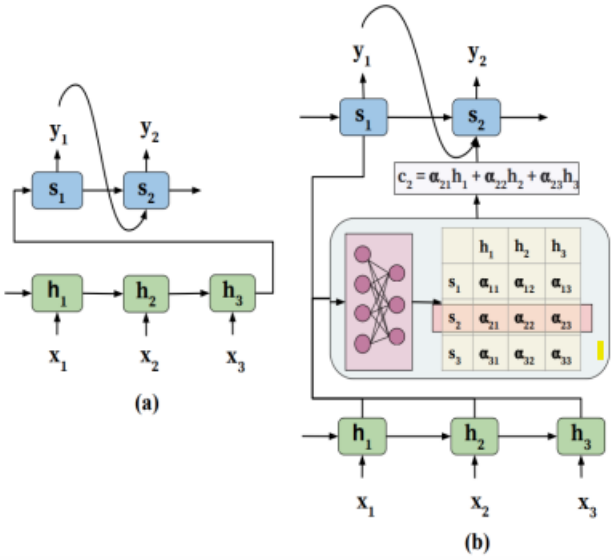


Figure 2.9. *Encoder-decoder architecture: (a) traditional (b) with attention model. Source: [3]*

The attention mechanism addresses these issues by allowing the decoder to access the entire encoded input sequence $\{h_1, h_2, \ldots, h_T\}$. It computes attention weights α over the input sequence to prioritize relevant positions for generating the next output token. The

encoder-decoder architecture with attention is depicted in Figure 2.9. The attention block learns attention weights α_{ij}, capturing the relevance between encoder hidden states h_i and decoder hidden states s_{j-1}. These weights help build a context vector c, which the decoder uses to generate the next output token. This context vector is a weighted sum of all encoder hidden states, improving alignment and output quality.

The attention model can be viewed as mapping a sequence of keys K to an attention distribution α according to a query q, where keys are encoder hidden states h_i and the query is the decoder hidden state s_{j-1}. The attention distribution α_{ij} emphasizes keys relevant to the query. The generalized attention model A works with key-value pairs (K, V) and query q:

$$A(q, K, V) = \sum_i p(a(k_i, q)) \cdot v_i \tag{2.56}$$

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

Table 2.1 compares the traditional encoder-decoder architecture with the attention model. Table 2.2 summarizes various alignment functions used in attention mechanisms.

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	$c = h_T$	$c_j = \sum_{i=1}^{T} \alpha_{ij} h_i$
Attention weights	-	$\alpha_{ij} = p(e_{ij})$
Alignment	-	$e_{ij} = a(s_{j-1}, h_i)$
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.*

2.5.5 Transformers

Transformers, introduced by Vaswani et al. [4], have revolutionized deep learning, particularly in natural language processing (NLP), computer vision (CV), and speech processing. Initially designed for sequence-to-sequence tasks in machine translation, Transformers have since demonstrated state-of-the-art performance across various domains. The adoption of Transformer-based pre-trained models (PTMs) has solidified their status as the preferred architecture in NLP, and their applications have extended to CV, audio processing, and other fields such as chemistry and life sciences.

The original Transformer model [4] is a sequence-to-sequence architecture comprising an encoder and a decoder, each consisting of multiple identical layers. Each encoder layer

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	$a(k_i, q) = \text{act}(q^T W k_i + b)$
generalized kernel	$a(k_i, q) = \phi(q)^T \phi(k_i)$
concat	$a(k_i, q) = w^T \text{imp}(W[q, k_i] + b)$
additive	$a(k_i, q) = w^T \text{imp}(W_1 q + W_2 k_i + b)$
deep	$a(k_i, q) = w^T \text{imp}(E_{L-1} + b_L)$
	$E(l) = \text{act}(W_l E_{l-1} + b_l)$
	$E(1) = \text{act}(W_1 k_i + W_0 q) + b_1$
location-based	$a(k_i, q) = a(q)$
feature-based	$a(k_i, q) = w^T \text{imp}(W_1 \phi_1(K) + W_2 \phi_2(K) + b)$

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, $W, w_{imp}, W_0, W_1, W_2$: trainable parameters, b: trainable bias term, act: activation function.

includes a multi-head self-attention mechanism and a position-wise feed-forward network (FFN), with residual connections [67] and Layer Normalization [68] applied around each module to enable deeper networks. The decoder layers, in addition to self-attention and FFN modules, incorporate cross-attention modules to focus on relevant parts of the input sequence. This architectural design is illustrated in Figure 2.10.

Attention Mechanisms

The Transformer utilizes an attention mechanism based on the Query-Key-Value (QKV) model. Given matrix representations of queries $Q \in \mathbb{R}^{N \times D_k}$, keys $K \in \mathbb{R}^{M \times D_k}$, and values $V \in \mathbb{R}^{M \times D_v}$, the scaled dot-product attention is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{D_k}}\right)V \tag{2.57}$$

where N and M are the lengths of the queries and keys (or values), D_k and D_v are the dimensions of the keys (or queries) and values, and the softmax function is applied row-wise. The scaling factor $\sqrt{D_k}$ mitigates the vanishing gradient problem associated with the softmax function.

Transformers use multi-head attention, projecting the original queries, keys, and values into multiple subspaces using different learned projections, enabling the model to focus on different parts of the input sequence. The multi-head attention mechanism is described by:

$$\text{MultiHeadAttn}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_H)W^O \tag{2.58}$$

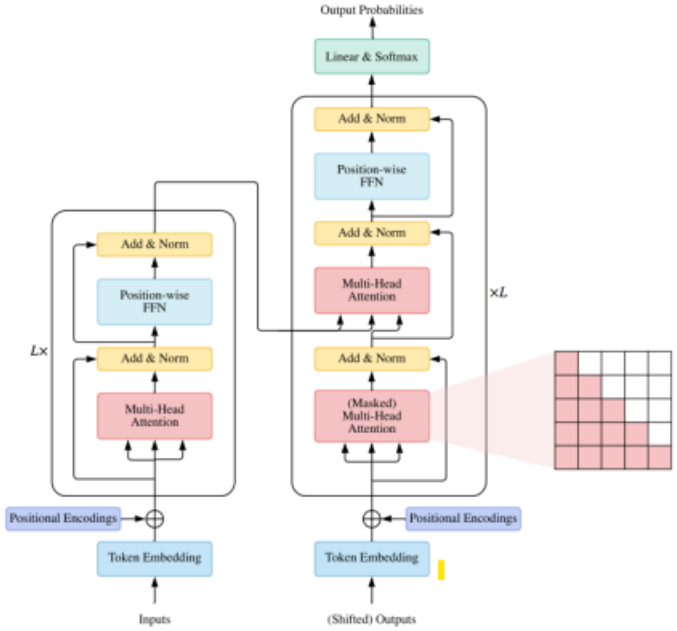


Figure 2.10. *Overview of vanilla Transformer architecture. Source: [4]*

where each head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.59}$$

This process is illustrated in Figure 2.11.

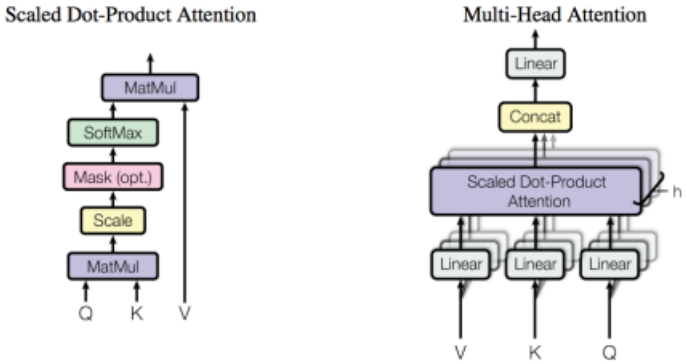


Figure 2.11. *(left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. Source: [4]*

The different types of attention mechanisms used in Transformers are as follows:

- **Self-attention**
  - $Q = K = V = X$
  - X is the output of the previous layer.
- **Masked Self-attention**
  - Queries at each position attend only to previous positions.

  - Prevents information leakage.

- **Cross-attention**
  - Queries are derived from the decoder layer.
  - Keys and values come from the encoder.

Position-wise Feed-Forward Networks (FFN)

The position-wise FFN applies a fully connected feed-forward network to each position separately and identically:

$$\text{FFN}(H') = \text{ReLU}(H'W_1 + b_1)W_2 + b_2 \tag{2.60}$$

where H' is the input, and $W_1 \in \mathbb{R}^{D_m \times D_f}$, $W_2 \in \mathbb{R}^{D_f \times D_m}$, $b_1 \in \mathbb{R}^{D_f}$, and $b_2 \in \mathbb{R}^{D_m}$ are trainable parameters. Typically, D_f is set to be larger than D_m.

2.5.6 Residual Connections and Normalization

To facilitate the training of deep models, residual connections [67] are used around each sublayer, followed by Layer Normalization [68]:

$$H' = \text{LayerNorm}(\text{SelfAttention}(X) + X) \tag{2.61}$$

$$H = \text{LayerNorm}(\text{FFN}(H') + H') \tag{2.62}$$

Positional Encoding

Transformers lack a mechanism to capture positional information since they do not inherently incorporate recurrence or convolution. Positional encodings are added to the input embeddings to provide information about the token positions:

$$\text{PE}_{(\text{pos},2i)} = \sin\left(\frac{\text{pos}}{10000^{2i/D_{\text{model}}}}\right) \tag{2.63}$$

$$\text{PE}_{(\text{pos},2i+1)} = \cos\left(\frac{\text{pos}}{10000^{2i/D_{\text{model}}}}\right) \tag{2.64}$$

where pos is the token position and i is the dimension. These encodings are added to the input embeddings to retain the positional context.

Table 2.3 presents the computational complexity and parameter counts of the core Transformer components.

Module	Complexity	Parameters
Self-attention	$O(T^2 \cdot D)$	$4D^2$
Position-wise FFN	$O(T \cdot D^2)$	$8D^2$

Table 2.3. *Complexity and Parameter Counts of Transformer Modules. Notation: T is the sequence length, D is the hidden dimension.*

Chapter 3

Natural Language Processing

3.1 Introduction

Natural languages have evolved in humans through use and repetition without conscious planning or premeditation, examples being English, Greek, and Latin. These languages differ 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 focused on the interactions between computers and human language. This definition is evolving due to the application of NLP in programming languages, source code modeling, and generation [69].

Natural language is unique because it is a discrete, symbolic, and categorical system specifically constructed to convey meaning. Unlike vision or other machine learning tasks where input data follow underlying physical laws, natural language consists of words that are symbols for extra-linguistic entities, where the word is a signifier mapping to a signified (idea or thing).

The hierarchical levels of language that NLP examines include:

- **Phonology:** This level deals with the interpretation of speech sounds within and across words. There are three types of rules used in phonological analysis: phonetic rules for sounds within words, phonemic rules for pronunciation variations when words are spoken together, and prosodic rules for stress and intonation fluctuations across a sentence.

- **Morphology:** This level handles the componential nature of words composed of morphemes, the smallest units of meaning.

- **Lexical:** At this level, both humans and NLP systems interpret the meaning of individual words, assigning part-of-speech tags based on context.

- **Syntactic:** This level analyzes the grammatical structure of sentences.

- **Semantic:** Semantic processing determines possible meanings by focusing on interactions among word-level meanings in a sentence, including the disambiguation of words with multiple senses.

- **Discourse:** This level works with text units longer than a sentence, focusing on text properties that convey meaning by connecting sentences.
- **Pragmatic:** This level explains how extra meaning is inferred from text without being explicitly encoded, requiring world knowledge including understanding of intentions, plans, and goals.

3.2 Applications

NLP provides theories and implementations for various applications. Any application utilizing text can benefit from NLP. Common applications include:

- **Information Retrieval and Web Search:** The science of searching for documents, information within documents, and metadata, as well as searching databases and the World Wide Web.
- **Information Extraction (IE):** The recognition, tagging, and extraction of critical information elements, such as persons, companies, locations, and organizations, from large text collections.
- **Text Summarization:** Distilling essential information from a source to produce an abridged version.
- **Question Answering (QA):** Extracting or generating answers from documents in response to questions.
- **Machine Translation (MT):** Using computer software to translate text or speech from one language to another.
- **Speech Recognition and Synthesis:** Extracting a textual representation from a spoken utterance.
- **Text Generation:** Generating sentences from "keywords".
- **Natural Language Understanding and Generation (NLU, NLG):** Converting a computer-based representation into a natural language representation.
- **Natural Language Inference (NLI):** 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 significantly dependent on input representation. In NLP, the input comprises natural language and its components, words. In English alone, there are an estimated 13 million words, presenting a vast array of possible inputs for a machine learning model. Ideally, word representation should encapsulate the word's meaning, encode properties such as similarity and difference between words, and distinguish polysemous words, which have different meanings in different contexts.

There are two main ways that linguists conceptualize meaning: "Denotational Semantics" and "Distributional Semantics." Denotational semantics represent an idea as a symbol (e.g., a word or a one-hot vector), while distributional semantics represent the meaning of a word based on its typical context.

3.3.1 Denotational Representation

In this category, words are mapped to symbols, either scalars or vectors.

Vocabulary IDs

Assume we have a vocabulary V for a given task, containing all the words in the training, development, and test datasets. The simplest approach is to assign each word in the vocabulary a unique ID. For example, for the vocabulary $V = \{\text{I, love, cats}\}$, we can define the following mapping:

$$w_{\text{I}} = 1,\ w_{\text{love}} = 2,\ w_{\text{cats}} = 3$$

This approach does not capture the word's meaning, nor does it address the notions of similarity and difference between words or the ambiguity problem. Additionally, this method is computationally expensive as it requires $|V|$ different IDs, potentially reaching up to 13 million. Finally, this approach offers no scope for further improvement.

One-Hot Encoding

One-hot encoding is similar to vocabulary IDs, where each word is represented by a vector. The main difference is the dimensionality augmentation of the subspace to allow further improvements through compression techniques. In one-hot encoding, each word is represented as a $\mathbb{R}^{|V|\times 1}$ vector with all zeros and a single one at the index corresponding to that word in a sorted list of the vocabulary. For example, word vectors in this encoding would be:

$$w_{\text{I}} = \begin{bmatrix}1\\0\\0\end{bmatrix},\quad w_{\text{love}} = \begin{bmatrix}0\\1\\0\end{bmatrix},\quad w_{\text{cats}} = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

In this representation, each word is treated as an independent symbol. This method does not inherently provide a notion of similarity or difference between words, nor does it solve the ambiguity problem. For instance:

$$w_{\text{cat}}^T w_{\text{dogs}} = w_{\text{siamese}}^T w_{\text{cat}} = 0$$

The resulting matrix representing this method will have dimensions $|V| \times |V|$ and appear as follows:

$$\begin{bmatrix}w_{\text{I}}\\w_{\text{love}}\\w_{\text{cats}}\end{bmatrix} = \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$

Given that this matrix is sparse, various compression techniques such as Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF) can be applied, but the final representation remains a form of "Denotational Semantics."

3.3.2 Distributional Semantics

As the British linguist J.R. Firth famously stated, "You shall know a word by the company it keeps," distributional semantics studies words based on their context. Word vectors in distributional semantics can be categorized into sparse and dense representations. Dense word vectors are commonly known as word embeddings.

Sparse Word Vectors

Term Frequency–Inverse Document Frequency (TF-IDF) Representation TF-IDF is a statistical measure used to evaluate the importance of a word t in a document d. A document can be defined as a batch of words. The TF-IDF value combines two terms: term frequency (tf) and inverse document frequency (idf). The tf term indicates that frequently occurring words are more significant, while the idf term suggests that words appearing too frequently are less important. Thus, the TF-IDF product balances these two terms.

The term frequency is given by:

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

The inverse document frequency is defined as:

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$$

where N is the total number of documents, and df_t is the number of documents in which term t appears.

The TF-IDF weighted value $w_{t,d}$ and the corresponding word vector is the product of these two terms:

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

Pointwise Mutual Information (PMI) Representation PMI, proposed by Fano [70], measures the association between two events x and y:

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

The PMI between a target word w and a context word c is defined as:

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

Positive PMI (PPMI) is used to replace negative PMI values with zero:

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

Given $|W|$ words and $|C|$ contexts, the PPMI matrix is:

$$\text{PPMI}_{ij} = \max\left(\log_2 \frac{p_{ij}}{p_{i*}p_{*j}}, 0\right)$$

Dense Word Vectors

Sparse vector representations, while informative, are computationally expensive and often less effective than dense representations. Dense vectors, or word embeddings, typically have dimensionalities ranging from 300 to 1200. They are more efficient and tend to perform better in models.

Word2Vec Word2Vec, developed by Mikolov et al. [5], is a two-layer neural network that learns vector representations of words by predicting word contexts. It has two architectures: Continuous Bag of Words (CBOW) and Skip-Gram.

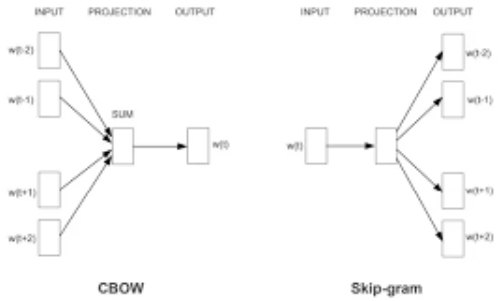


Figure 3.1. *The CBOW architecture predicts the current word based on the context, while Skip-gram predicts surrounding words given the current word. Source: [5]*

GloVe Embeddings GloVe, proposed by Pennington et al. [71], is an unsupervised learning algorithm for obtaining vector representations by aggregating global word-word co-occurrence statistics from a corpus. The training objective is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence.

Contextual Embeddings Static embeddings like Word2Vec and GloVe cannot handle polysemous words effectively. Contextual embeddings, such as ELMo [72] and BERT [7], address this by learning dynamic representations where the vector for each word changes depending on its context.

3.4 Language Models

The goal of a Language Model (LM) is to estimate the likelihood of a sequence of words. For instance, it should recognize that the sentence "I love cats." is more probable than "Cats book ice-cream," based on both syntax and semantics. Formally, a language model calculates the probability of a sequence of words as follows:

$$P(w_1, w_2, \ldots, w_n) \tag{3.1}$$

This probability can be decomposed as:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1}, w_{i-2}, \ldots, w_1) = P(w_1)P(w_2 \mid w_1)\ldots P(w_n \mid w_1, \ldots, w_{n-1}) \tag{3.2}$$

3.4.1 Traditional Language Models

Calculating the probability of a sentence using the above formula, known as an N-Gram LM, can be computationally intensive. To make this process more efficient, certain assumptions are made during training.

The simplest assumption is the unigram model, where word occurrences are considered completely independent:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i) \tag{3.3}$$

For training a unigram model with a corpus C containing $|N|$ words, the probability of each word w_i is given by:

$$P(w_i) = \frac{\text{count}(w_i)}{|N|} \tag{3.4}$$

where $\text{count}(w_i)$ is the number of occurrences of word w_i in the training corpus. However, this model does not account for the dependency between words.

To develop a more accurate LM, we utilize Bayesian probability theory and the Markov assumption, which states that a word depends only on a fixed number of preceding words. This leads to Bigram and Trigram models.

A Bigram LM assumes each word depends only on the previous word:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=2}^{n} P(w_i \mid w_{i-1}) \tag{3.5}$$

The training formula for a Bigram model is:

$$P(w_i \mid w_{i-1}) = \frac{\text{count}(w_i, w_{i-1})}{\sum_{w \in C} \text{count}(w_{i-1}, w)} \tag{3.6}$$

A Trigram LM extends this by considering the two preceding words:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=3}^{n} P(w_i \mid w_{i-1}, w_{i-2}) \tag{3.7}$$

The training formula for a Trigram model is:

$$P(w_i \mid w_{i-1}, w_{i-2}) = \frac{\text{count}(w_i, w_{i-1}, w_{i-2})}{\sum_{w \in C} \text{count}(w_{i-2}, w_{i-1}, w)} \tag{3.8}$$

The choice of n in n-gram models typically depends on the amount of available training data. Trigram models, which use a two-word history, are common. However, higher-order n-grams can be used if sufficient training data is available.

3.4.2 Neural Language Models

Non-linear neural network models enable conditioning on increasingly large context sizes with only a linear increase in the number of parameters. A notable example is the neural probabilistic language model popularized by [6]. This model uses vector representations of a word window of n previous words, looked up in a table C, known as word embeddings. These word embeddings, $C(w) \in \mathbb{R}^{d_w}$, are concatenated and fed into a hidden layer, whose output is provided to a softmax layer, as shown in Figure 3.2. The mathematical formulation of this neural language model is as follows:

$$x = [C(w_1); C(w_2); \ldots; C(w_n)] \tag{3.9}$$
$$\hat{y} = P(w_i \mid w_{1:k}) = LM(w_{1:k}) = \text{softmax}(hW_2 + b_2) \tag{3.10}$$
$$h = g(xW_1 + b_1) \tag{3.11}$$
$$x = [C(w_1); C(w_2); \ldots; C(w_n)] \tag{3.12}$$
$$C(w) = E[w] \tag{3.13}$$

where $w_i \in V$, $E \in \mathbb{R}^{|V| \times d_w}$, $W_1 \in \mathbb{R}^{n \cdot d_w \times d_{hid}}$, $b_1 \in \mathbb{R}^{d_{hid}}$, $W_2 \in \mathbb{R}^{d_{hid} \times |V|}$, $b_2 \in \mathbb{R}^{|V|}$. V is a finite vocabulary. The vocabulary size $|V|$ ranges between 1,000 and 1,000,000 words, with the common size being around 70,000 unique words. Feedforward neural networks have since been replaced with recurrent neural networks [73] for language modeling.

3.5 Embeddings from Language Models (ELMo)

ELMo embeddings [**?**] are deep contextualized word representations that provide high-quality language modeling by capturing complex characteristics of word use and adjusting them in various linguistic contexts. These vectors are derived from a bi-directional LSTM trained with a coupled language model (LM) objective on a large text corpus. ELMo representations are functions of all internal layers of the bi-directional language model. However, the weighting of the ELMo embeddings needs careful tuning for each specific task, posing some limitations despite their effectiveness.

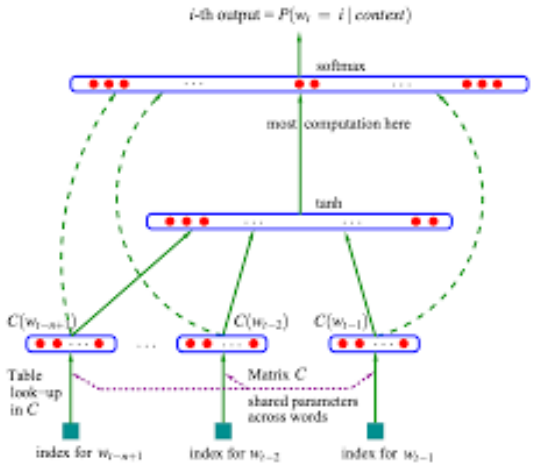


Figure 3.2. *A feed-forward neural network language model. Source: [6]*

3.6 Bidirectional Encoder Representations from Transformers (BERT)

BERT [7] proposed by J. Devlin et al. is a novel approach to incorporate bidirectionality in a single Transformer model. Direct approaches to incorporating bidirectionality in Transformer models are challenging because they allow words to see themselves in the context from multiple layers, making it impossible to use as a Language Model. Traditionally, only unidirectional encoders—either left-right or right-left models—could be trained. However, bidirectional models, which could see the complete sequence context, are inherently more powerful than unidirectional models or concatenations of two unidirectional models. To address this, the authors trained their model on two unsupervised prediction tasks:

Masked Language Model: To overcome the challenges of applying bidirectionality in Transformers, [7] proposed masking random tokens in the sequence. The Transformer is trained to predict only the masked words while being able to view the whole sequence. WordPiece Tokenization generates the sequence of tokens, where rare words are split into sub-tokens. Then, 15% of the WordPiece Tokens are masked. The masking approach is as follows:

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

Predicting only 15% of the words instead of all words makes BERT slower to converge. However, BERT showed immediate improvements in absolute accuracy while converging slightly slower than traditional unidirectional models.

Next Sentence Prediction: This task involves predicting whether the first sequence immediately precedes the next. This helps the Transformer perform better on tasks such as

question-answering and natural language inference, which involve understanding relationships between input sequences. The dataset for training had a balanced 50/50 distribution created by choosing actual pairs of neighboring sentences for positive examples and a random second sentence for negative examples. The input sequence for this task is:

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

where sentences A and B are two sentences after performing the masking operations. The [CLS] token is used for obtaining a fixed vector representation for classification, and [SEP] separates the two input sequences. This method achieved an impressive accuracy of 97–98% in the next sentence prediction task.

Pre-Training Procedure: The authors used the BooksCorpus and English Wikipedia for pretraining data, with two variations: BERT-BASE (12 layers) and BERT-LARGE (24 layers). The maximum input sequence length is 512 tokens. A dropout value of 0.1 is used for regularization. GELU (Gaussian Error Linear Units) is used instead of ReLU for activation functions, providing improvements. Training was performed on TPUs: BERT-BASE on 16 TPU chips for four days and BERT-LARGE on 64 TPU chips for four days.

Fine-Tuning Procedure: The pre-trained BERT can be fine-tuned on a relatively small dataset, requiring less processing power, as shown in Figure 3.3. BERT improved upon the previous state-of-the-art in tasks such as natural language inference, question answering, semantic similarity, and linguistic acceptability. The tasks can be broadly divided into:

- Single Sentence Classification Tasks: Performed by adding layers on the [CLS] token 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 is performed by adding layers to the [CLS] token.
- Question Answering Tasks
- Single Sentence Tagging Tasks

Two multilingual BERT models (uncased and cased) for over 102 languages were also released. Furthermore, OpenAI released GPT2 [74], which is essentially BERT trained as a language model on a considerable amount of data.

Recent advancements have continued to build on BERT's architecture. For instance, RoBERTa [75] optimizes the pre-training process by training with more data and longer sequences, and ELECTRA [76] introduces a novel pre-training task that trains the model to distinguish between real and fake tokens generated by another model, making the training more efficient. Moreover, ALBERT [77] reduces the number of parameters significantly by sharing parameters across layers and decomposing the embedding matrix, which improves training speed and lowers memory consumption without sacrificing performance.

Mathematical Notation: Attention is a core component of Transformers, consisting of several layers, each containing multiple attention heads. Each attention head gathers

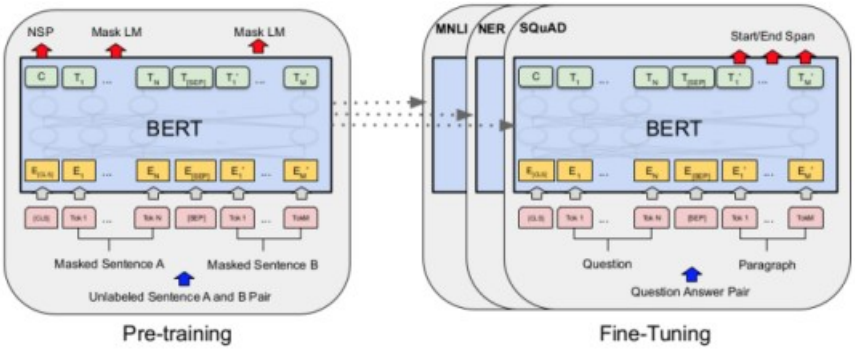


Figure 3.3. *Pre-training and fine-tuning procedures for BERT. Source: [7]*

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 $y_i \in \mathbb{R}^d$ from the corresponding pre-update vector $\tilde{y}_i \in \mathbb{R}^d$ and a sequence of input vectors $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$:

$$y_i = \left(\sum_{j=1}^{n} \alpha_{i,j} v(x_j) \right) W_O$$

$$\alpha_{i,j} := \mathrm{softmax}_{x_j \in X} \left(\frac{q(\tilde{y}_i) k(x_j)^\top}{\sqrt{d_0}} \right) \in \mathbb{R}$$

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

$$q(\tilde{y}_i) := \tilde{y}_i W_Q + b_Q \quad \left(W_Q \in \mathbb{R}^{d \times d_0}, b_Q \in \mathbb{R}^{d_0} \right)$$

$$k(x_j) := x_j W_K + b_K \quad \left(W_K \in \mathbb{R}^{d \times d_0}, b_K \in \mathbb{R}^{d_0} \right)$$

$$v(x_j) := x_j W_V + b_V \quad \left(W_V \in \mathbb{R}^{d \times d_0}, b_V \in \mathbb{R}^{d_0} \right)$$

Attention gathers value vectors $v(x_j)$ based on attention weights and then applies matrix multiplication $W_O \in \mathbb{R}^{d_0 \times d}$. Boldface letters such as x denote row (not column) vectors, following the notations in Vaswani et al.. In self-attention, the input vectors X and the pre-update vector $\tilde{y}_i$ are previous layer's output representations. In source-target attention, X corresponds to the encoder representations, and vector $\tilde{y}_i$ (and updated vector 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 process language in a way that is not exclusive to a single task, genre, or dataset. The General Language Understanding Evaluation (GLUE) benchmark [8] is a collection of tools for evaluating model performance across a diverse set of NLU tasks. The GLUE benchmark

includes the following tasks and datasets:

1. Single-Sentence Task
    - **CoLA:** Corpus of Linguistic Acceptability [37]
    - **SST-2:** Stanford Sentiment Treebank [36]
2. Similarity and Paraphrase Tasks
    - **MRPC:** Microsoft Research Paraphrase Corpus [35]
    - **QQP:** Quora Question Pairs
    - **STS-B:** Semantic Textual Similarity Benchmark [78]
3. Inference Tasks
    - **MNLI:** Multi-Genre Natural Language Inference [33]
    - **QNLI:** Question-answering Natural Language Inference [34]
    - **RTE:** Recognizing Textual Entailment
    - **WNLI:** Winograd Natural Language Inference [38]

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: [8]*

3.7.1 CoLA

The Corpus of Linguistic Acceptability [37] 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 as the evaluation metric, which evaluates performance on unbalanced binary classification and ranges from -1 to 1, with 0 being the performance of uninformed guessing.

3.7.2 SST-2

The Stanford Sentiment Treebank [36] consists of sentences from movie reviews with 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 only sentence-level labels.

3.7.3 MRPC

The Microsoft Research Paraphrase Corpus [35] 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 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. The class distribution in QQP is unbalanced (63 percent negative), so GLUE reports both accuracy and F1 score.

3.7.5 STS-B

The Semantic Textual Similarity Benchmark [78] 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. GLUE evaluates using Pearson and Spearman correlation coefficients.

3.7.6 MNLI

The Multi-Genre Natural Language Inference Corpus [33] 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.

3.7.7 QNLI

The Stanford Question Answering Natural Language Inference Dataset [34] 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. 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.

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 [38] 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.

Chapter 4

Compression of Deep Learning Models

4.1 Introduction

Transformers have revolutionized the field of Natural Language Processing (NLP), achieving remarkable performance across a variety of tasks. Despite their success, training these models is computationally intensive, and they often contain millions of parameters. The base BERT model, for instance, has 110 million parameters, but more recent models like GPT-4 boast even more parameters (OpenAI, 2023). The development continues with models like Google's Gemini and Anthropic's Claude, which further push the limits of size and capability(hundreds of billions parameters). This rapid growth has sparked concerns about computational complexity, environmental impact, and fairness in comparing different architectures, as well as reproducibility [43][44][45]. This exponential growth raises several concerns, including the computational complexity of self-attention mechanisms, environmental impacts [43, 79], fair comparisons of different architectures [80], and issues of reproducibility.

The intricacies of human language may indeed require an extensive number of parameters for comprehensive modeling, but current models do not optimally utilize their parameters. For instance, Voita et al. [11] demonstrated that the majority of Transformer heads can be pruned without significantly affecting performance. Similarly, Clark et al. [46] found that most heads within the same layer exhibit similar self-attention patterns, which likely explains why Michel et al. [17] were able to consolidate many layers down to a single head.

Depending on the task, some heads and layers in BERT are not only superfluous [81], but can also degrade downstream task performance. The beneficial effects of disabling certain heads have been observed in machine translation [17], abstractive summarization [82], and GLUE tasks [10]. Additionally, Tenney et al. [48] noted that in 5 out of 8 probing tasks, some layers led to decreased performance scores. From the perspective of unstructured pruning, Gordon et al. [49] discovered that 30–40% of the weights could be pruned without impacting downstream tasks.

Generally, larger BERT models perform better [83, 84], but this is not universally true: BERT-base outperformed BERT-large on tasks such as subject-verb agreement [50] and sentence subject detection [85]. Clark et al. [46] suggested that one reason for the redundancy might be the use of attention dropouts, which zero out some attention weights

during training.

Given the evidence of overparameterization, it is not surprising that BERT can be effectively compressed with minimal accuracy loss, making it highly attractive for practical applications.

4.2 Compression: Problem Setting

The aim of model compression is to reduce the size of a model without significantly compromising its accuracy. The compressed model should have fewer and smaller parameters, thereby using less RAM at runtime and reducing latency. This is beneficial as it frees up memory for other applications and decreases energy consumption during runtime. Larger models typically require more memory accesses, leading to increased latency. Common methods for compressing neural models include:

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

4.2.1 Pruning

Pruning involves the removal of connections between neurons or entire neurons, channels, or filters from a trained network by zeroing out values in its weight matrix or eliminating groups of weights entirely. For instance, to prune a single connection, a weight in the matrix is set to zero; to prune a neuron, all values in a column are set to zero. The motivation for pruning is that networks often have redundant features. Pruning can be categorized into unstructured pruning, which removes individual weights or neurons, and structured pruning, which removes entire channels or filters. Both types differ in implementation and results.

Unstructured Pruning By setting certain weights in a matrix to zero, unstructured pruning increases the network's sparsity. Depending on the degree of sparsity and the storage method, pruned networks can occupy much less memory than dense ones.

How do we decide which weights to prune? One widely used method, magnitude-based pruning, compares the magnitudes of weights to a threshold. Han et al.'s 2015 paper [86] popularized this approach. Pruning is applied layer-by-layer; a "quality parameter" multiplied by the standard deviation of weights in a layer sets the threshold, below which weights are zeroed. After pruning, the model is retrained to adjust the remaining weights, and this process is repeated.

Mathematical formalism of Structured Pruning Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$ and a desired sparsity level κ (number of non-zero weights), neural network weight pruning

can be expressed as the following constrained optimization problem:

$$\min_{w} L(w; D) = \min_{w} \frac{1}{n} \sum_{i=1}^{n} \ell(w; (x_i, y_i)) \text{s.t. } w \in \mathbb{R}^m, \|w\|_0 \leq \kappa \tag{4.1}$$

Here, $\ell(\cdot)$ is the standard loss function (e.g., cross-entropy loss), 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. Minimizing the L_0 norm is non-convex, NP-hard, and requires combinatorial search.

Structured Pruning Unlike unstructured pruning, structured pruning removes entire blocks of weights within given matrices, avoiding problematic sparse connectivity patterns. This allows the pruned model to run using the same hardware and software as the original. Groups of weights can be ranked according to their L_1 norms for pruning. More sophisticated, data-driven approaches have also been proposed for better results.

Huang et al. [87] were the first to integrate performance-size tradeoff control into the pruning process. Their algorithm produces a set of "pruning agents," each corresponding to a convolutional layer of the network, and an alternative pruned version of the original model. Pruning agents maximize an objective parametrized by a "drop bound" value, maintaining performance above a specified level. Pruning agents for each convolutional layer evaluate the effects of pruning combinations of different filters. The alternative model's performance on an evaluation set is compared to that of the original, learning modifications that improve efficiency while adhering to accuracy constraints. Once trained, the alternative model is retrained to adjust for changes, and the process repeats for the next layer.

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

$$\min_{w_s} L(w_s; D) = \min_{w_s} \frac{1}{n} \sum_{i=1}^{n} \ell(w_s; (x_i, y_i)) \text{s.t. } w_s \in \mathbb{R}^m, \|w_s\|_0 \leq \kappa \tag{4.2}$$

Here, $\ell(\cdot)$ is the standard loss function (e.g., cross-entropy loss), w_s is the structural set of parameters (e.g., a convolutional filter), m is the total number of structural sets, and $\|\cdot\|_0$ is the standard L_0 norm. Minimizing the L_0 norm is non-convex, NP-hard, and requires combinatorial search.

Accelerating unstructured sparse matrix multiplication is a field of active research with recent progress. For instance, bank-balanced sparsity, closely related to unstructured sparsity, achieves near-ideal speed-ups with minimal deviation from unstructured sparsity [88]. Adaptive sparse matrix multiplication has shown promising results on GPUs, though not yet on silicon.

4.2.2 Quantization

Quantization compresses models by reducing the size of weights. Generally, it maps values from a large set to a smaller set, reducing the range of possible output values without significant information loss. In neural networks, weights or activation outputs of a layer tend to be normally distributed within a specific range. An ideal quantization schema adapts to fit each layer's distribution. For example, reducing 32-bit floating-point numbers to 8-bit fixed points, which has 256 possible values, can cut the memory footprint by a quarter.

4.3 Lottery Ticket Hypothesis (LTH)

The pruning pipeline involves several stages: initially, the entire network is trained, followed by pruning the trained network using a specific pruning algorithm. A critical question arises: Can the pruned network be trained from scratch to achieve comparable or even superior performance? This is generally not possible unless the network is initialized appropriately. Frankle and Carbin [9] proposed that effective initialization is the original weights before the training-pruning process.

The proposed pipeline is as follows: train the initial model, apply a pruning algorithm to identify the pruned network, and then retrain the network by initializing the remaining connections to their original weights. This concept, known as the Lottery Ticket Hypothesis, posits that a randomly initialized, dense neural network contains a subnetwork that, when trained in isolation, can achieve the original network's accuracy within a comparable number of iterations [9].

Frankle and Carbin not only introduced this pruning pipeline but also provided insights into the functioning of deep neural networks. Overparameterization in these networks leads to the creation of many subnetworks, some of which can perform competitively on their own. These high-performing subnetworks are termed "Winning Tickets." These winning tickets have been shown to generalize across vision datasets [89] and are present in both LSTM and Transformer models for natural language processing (NLP) [90].

To identify winning tickets, Frankle et al. [9] employed the simplest unstructured pruning method: training a network and pruning the smallest-magnitude weights. The remaining connections form the architecture of the winning ticket. Each pruned connection is then reset to its initial value from the original network before training. The process can be described as follows:

1. Randomly initialize a neural network $f(x; \theta_0)$ where $\theta_0 \sim D_\theta$.
2. Train the network for j iterations, resulting in 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, forming the winning ticket $f(x; m \odot \theta_0)$.

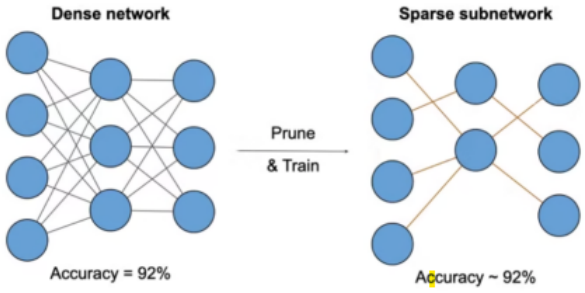


Figure 4.1. *Graphic Illustration of Lottery Ticket Hypothesis*

This one-shot pruning approach involves training the network once, pruning $p\%$ of the weights, and resetting the surviving weights. However, Frankle et al. focused on iterative pruning (IP), which repeatedly trains, prunes, and resets the network over several rounds. Each round prunes $\frac{p}{n}\%$ of the remaining weights, allowing iterative pruning to find winning tickets that match the original network's accuracy at smaller sizes compared to one-shot pruning.

Figure 4.2 shows the results of applying the lottery ticket hypothesis to a LeNet-300-100 architecture [91] trained on MNIST. The pruned model's performance across various pruning rates surpasses that of the entire network.

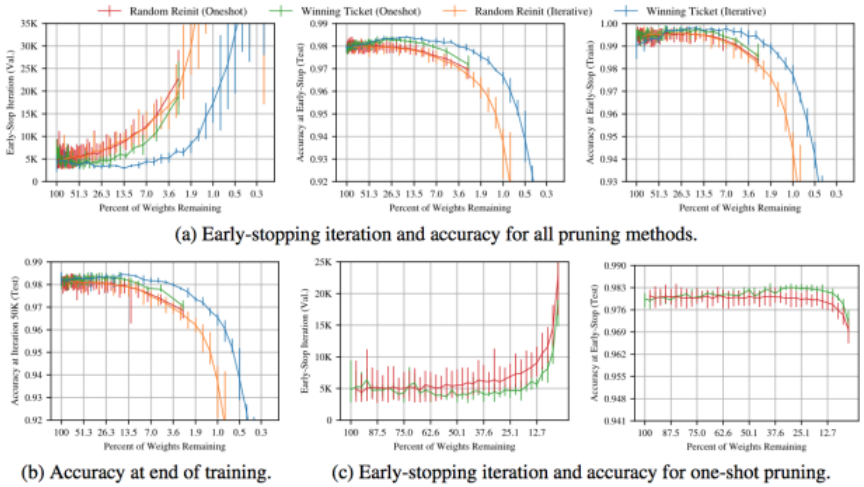


Figure 4.2. *Early-stopping iteration and accuracy of LeNet under one-shot and iterative pruning. Average of five trials; error bars indicate the minimum and maximum values. Source: [9]*

4.4 Pruning Transformer-based models

4.4.1 Transformer-based Structured Pruning

BERTology, the field of study focused on understanding the intricacies of large-scale transformers like BERT, has led to various pruning techniques based on insights from BERT's internal workings [51]. Kovaleva et al. [10] identified a limited set of repetitive

attention patterns across different heads, highlighting the model's overparameterization. They demonstrated that disabling specific attention heads could enhance performance in fine-tuned BERT models.

Their primary tool was self-attention maps, which are 2D float arrays representing self-attention weights for each head in every layer for a given input sequence. They identified several self-attention patterns in BERT, as shown in Figure 4.3:

- **Vertical**: Attention to special BERT tokens [CLS] and [SEP], which serve as delimiters between individual chunks of BERT's inputs.
- **Diagonal**: Attention to the previous/following tokens.
- **Vertical and Diagonal**: A mix of the previous two types.
- **Block**: Intra-sentence attention for tasks with two distinct sentences, such as RTE or MRPC.
- **Heterogeneous**: Highly variable depending on the specific input and not characterized by a distinct structure.

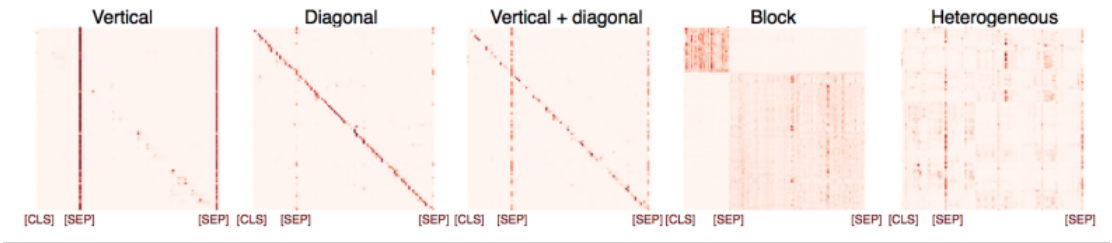


Figure 4.3. *Typical self-attention classes used for training a neural network. Both axes on every image represent BERT tokens of an input example, and colors denote absolute attention weights (darker colors indicate greater weights). The first three types are most likely associated with language model pre-training, while the last two potentially encode semantic and syntactic information. Source: [10]*

Their findings suggested that disabling some heads could improve performance across tasks, while others might be detrimental. Importantly, disabling certain heads consistently across tasks and datasets led to performance improvements.

Additionally, Voita et al. [11] analyzed the role of individual attention heads in the Transformer model for neural machine translation. They used Layer-wise Relevance Propagation (LRP) [92] to evaluate the contribution of different heads at each layer to the model's predictions. They found that a small number of heads in each layer were significantly more important than others, as shown in Figure 4.4.

They categorized heads into three main types based on their attention patterns:

- **Positional**: Points to an adjacent token.

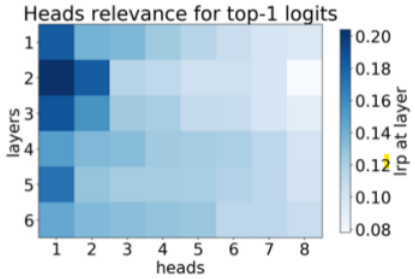


Figure 4.4. *Importance (according to LRP) of self-attention heads. The model trained on 6m OpenSubtitles EN-RU data. Source: [11]*

- **Syntactic**: Points to tokens in a specific syntactic relation.
- **Rare Words**: Points to the least frequent tokens in a sentence.

Voita et al. also proposed a structured pruning methodology by gating heads [11]. They modified the Transformer architecture by introducing scalar gates g_i for each head i:

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

These gates are specific to heads and independent of the input. The goal was to disable less critical heads entirely rather than simply downweighing them, ideally using L0 regularization:

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

Since the L0 norm is non-differentiable, they used a stochastic relaxation with Hard Concrete distributions [93].

Michel et al. [17] conducted comprehensive ablation experiments to understand the impact of individual attention heads on model performance. By modifying the multi-head attention formula and using mask variables, they determined the sensitivity of the model to the removal of each head. Their approach can be described with the following equations. The modified multi-head attention is given by:

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

where ξ_h are mask variables with values in $\{0, 1\}$. The importance of a head h is measured by the expected sensitivity of the model to the mask variable ξ_h:

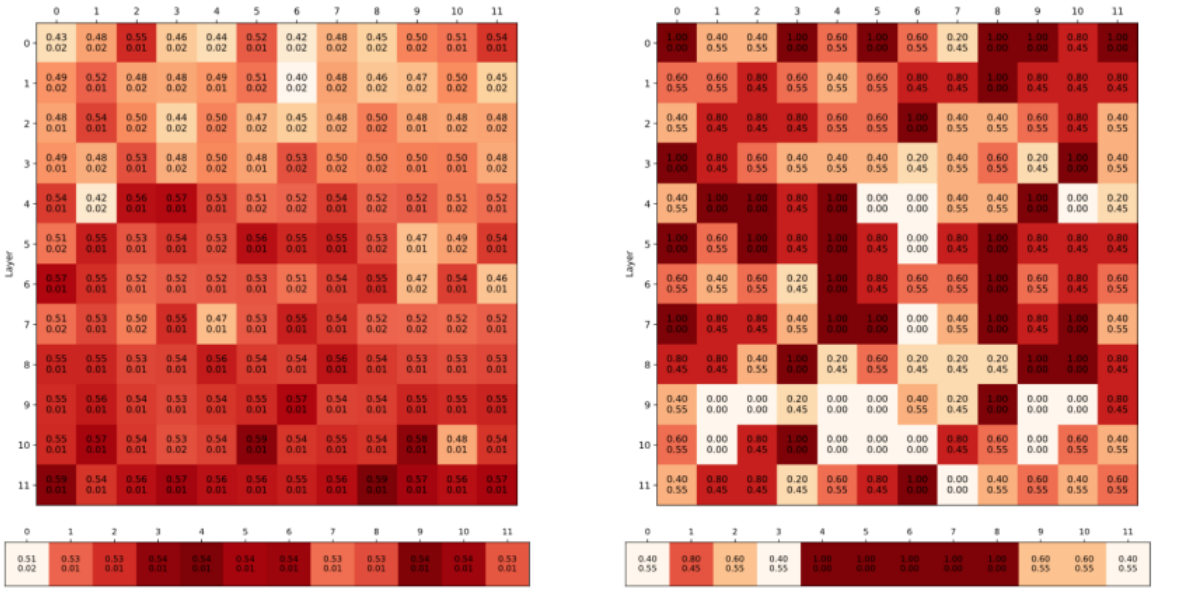
$$I_h = \mathbb{E}_{x \sim X} \left| \frac{\partial L(x)}{\partial \xi_h} \right|$$

where X is the data distribution and $L(x)$ is the loss on sample x. This can be expanded using the chain rule:

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

Their findings revealed that a significant proportion of attention heads could be removed with minimal impact on performance, suggesting that models can operate effectively with far fewer heads than originally designed. This insight is crucial for optimizing and compressing large models, as it points to potential areas of redundancy.

Prasanna et al. [12] expanded on these ideas by applying both magnitude and structural pruning to fine-tuned BERT models. They used an iterative approach to prune the lowest magnitude weights and evaluate the importance of self-attention heads and MLPs. Their results showed patterns in the pruned heads for various tasks, such as QNLI, and identified "good" and "bad" subnetworks within BERT. Notably, they found that the pruned subnetworks, when retrained, could achieve performance close to the original model. This work underscores the potential of pruning as a method to streamline models without sacrificing accuracy.



(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 4.5. *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: [12]*

Prasanna et al. also explored the overlaps in BERT's "good" subnetworks across GLUE tasks, indicating shared heuristics or patterns encoded in combinations of BERT elements.

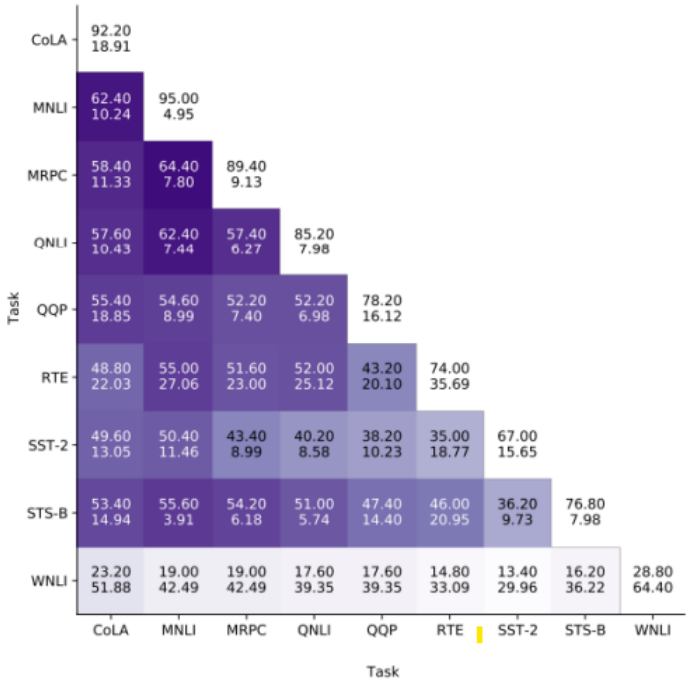


Figure 4.6. *Overlaps in BERT's "good" subnetworks between GLUE tasks: self-attention heads. Source: [12]*

4.4.2 Transformer-based Magnitude Pruning

Chen et al. [13] applied the unstructured Lottery Ticket Hypothesis on pretrained BERT models. They investigated the accuracy of training subnetworks of neural networks. For a network $f(x;\theta,\cdot)$, a subnetwork is defined as $f(x;m\odot\theta,\cdot)$ with a pruning mask $m\in\{0,1\}^{d1}$ (where $\odot$ represents the element-wise product), effectively setting some weights to zero.

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

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

$$\epsilon^T(A_t^T(f(x;m\odot\theta,\gamma)))\geq\epsilon^T(A_t^T(f(x;\theta_0,\gamma)))$$

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

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

To identify subnetworks $f(x;m\odot\theta,\cdot)$, Chen et al. employed neural network pruning

[13, 14]. They determined the pruning mask m by training the unpruned network to completion on a task T (i.e., using A_t^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 instance, 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 iterative pruning is more effective. The Iterative Magnitude Pruning (IMP) process is described below.

Αλγοριθμος 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: $f(x; \theta_i, \gamma_i) = A_i^T(f(x; \theta_0, \gamma_0))$.
2: Set the initial pruning mask to $m = \mathbf{1}^{d1}$.
3: **repeat**
4: Train $f(x; m \odot \theta_i, \gamma_i)$ to step t: $f(x; m \odot \theta_t, \gamma_t) = A_{t-i}^T(f(x; m \odot \theta_i, \gamma_i))$.
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, depicted in Figure 4.7, indicate that despite BERT being a pre-trained language model, the Lottery Ticket Hypothesis can be realized 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 4.7. *Performance of subnetworks at the highest sparsity for which IMP finds winning tickets on each task. To account for fluctuations, a subnetwork is considered 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: [13]*

4.5 Pruning Computer Vision Models

In the literature, there is a notable exchange of ideas between Natural Language Processing (NLP) and Computer Vision (CV). This section explores some concepts related to pruning models in the CV domain.

Mallya et al. [14] introduced PackNet, an approach that iteratively prunes unimportant weights and fine-tunes the model for new tasks. The result of pruning and weight modifications is a binary parameter usage mask, as illustrated in Figure 4.8.

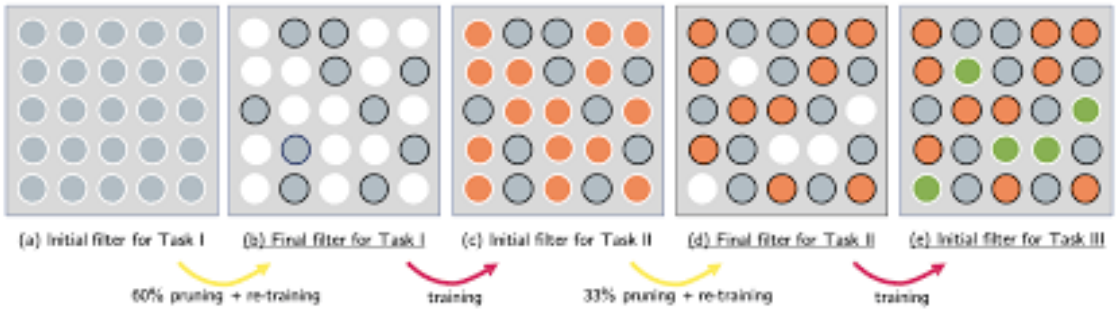


Figure 4.8. *Illustration of the evolution of a 5 × 5 filter with steps of training. Initial training of the network for Task I learns a dense filter as illustrated in (a). After pruning by 60% and re-training, a sparse filter for Task I is obtained, as depicted in (b), where white circles denote 0 valued 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: [14]*

Another approach, "Piggyback", proposed by Mallya et al. [14], does not alter the initial backbone network weights but instead learns a different mask for each task. This method is agnostic to task order, and the addition of new tasks does not affect performance on previous tasks. This is depicted in Figure 4.9.

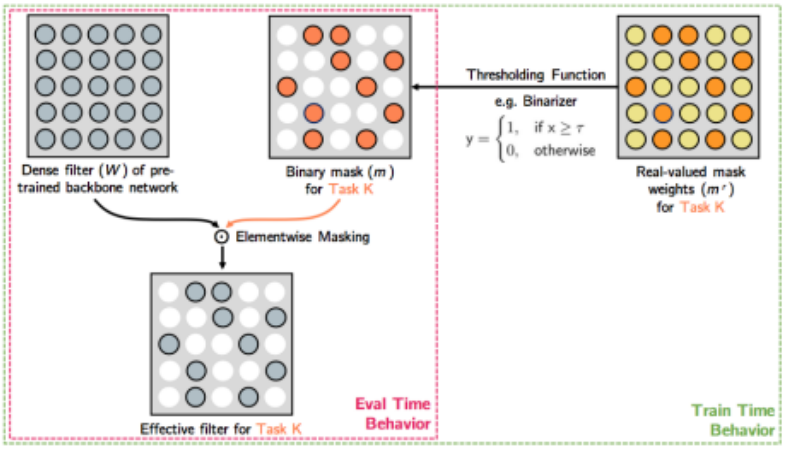


Figure 4.9. *Overview of Piggyback method for learning piggyback masks for fixed backbone networks. During training, a set of real-valued weights m_r are maintained, which are passed through a thresholding function to obtain binary-valued masks m. These masks are applied to the weights W of the backbone 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: [14]*

Chapter 5

Interpretability

5.1 Introduction

The remarkable success of Transformer-based models can be largely attributed to their powerful multi-head self-attention mechanism, which effectively learns token dependencies and encodes contextual information from the input. However, the complexity of these models has raised concerns regarding their interpretability and transparency. As these models are increasingly deployed in critical applications, understanding their decision-making process has become a crucial aspect of AI research.

Interpretability techniques aim to shed light on the inner workings of machine learning models, enabling us to attribute model decisions to individual input features and understand the influence of various components of the model. These techniques can be broadly categorized into two types: intrinsic and post-hoc interpretability. Intrinsic interpretability involves designing models that are inherently interpretable, such as decision trees or linear models, where the relationship between input features and predictions is explicit. Post-hoc interpretability, on the other hand, involves applying techniques to understand and explain the predictions of complex models, such as neural networks, after they have been trained.

One fundamental approach to interpretability is attribution, which refers to techniques that assign credit or importance to individual input features (such as words in a sentence or pixels in an image) to explain how they influence the model's prediction. Attribution techniques seek to answer the question: "Which parts of the input contributed most to the model's output?" By quantifying the importance of input features, attribution methods offer a window into the internal workings of complex models, helping to demystify their predictions.

Attribution methods come in various forms, but all share the same goal: assigning importance to input features in relation to the final prediction. In natural language processing (NLP), for example, the input features are often words or tokens in a sentence, and attribution techniques aim to highlight which words had the most impact on the model's decision.

Several interpretability methods have been developed to address this need, each with its own strengths and limitations. Saliency-based methods, such as Grad-CAM [94] and Integrated Gradients [22], highlight the importance of input features by computing gradients or similar measures. Perturbation-based methods, such as LIME [95] and SHAP

[96], explain model predictions by observing the changes in output when input features are perturbed. Contextual decomposition methods, such as Contextual Decomposition (CD) [97] and Agglomerative Contextual Decomposition (ACD) [98], analyze the contributions of different input features or groups of features to the model's predictions.

One of the most widely used methods for interpretability is Integrated Gradients (IG). IG addresses the limitations of classic gradient methods by satisfying two fundamental axioms: (a) Implementation Invariance, and (b) Sensitivity. These properties make IG a reliable and useful method for feature attribution. In this thesis, I propose ideas centered around IG, which are the subject of ongoing research. First, I introduce various techniques that I studied before concluding with IG, highlighting their application, particularly in the context of language models.

The following sections delve into classic methods for neural network interpretability and their application to language models, exploring gradient-based approaches, external explainers based on model behaviors, and contextual decomposition techniques.

5.2 Classic Methods

5.2.1 Gradient Based

A lot of papers have emerged about interpretability and explainability based on gradients of the output or intermediate neurons. Selvaraju et al. [94] introduced Grad-CAM, which computes a coarse-grained feature-importance map by associating the feature maps in the final convolutional layer with particular classes based on the gradients of each class with respect to each feature map, and then using the weighted activations of the feature maps as an indication of which inputs are most important.

The authors in [99] presented DeepLIFT, a method that compares the activation of each neuron to its 'reference activation' and assigns contribution scores $C_{\Delta x_i \Delta t}$ according to the difference. $C_{\Delta x_i \Delta t}$ can be thought of as the amount of difference-from-reference in t that is attributed to or 'blamed' on the difference-from-reference of x_i. It is a back-propagation approach, contrasting forward propagation methods that propagate an importance signal from an output neuron backward through the layers to the input in one pass.

The latter method overcomes the limitations of classic gradient methods like Integrated Gradients [22] and Gradient x Input [100], as a neuron can signal meaningful information even in the regime where its gradient is zero. All of these methods require a reference input vector (e.g., Gradient x Input uses a reference of all zeros, while Integrated Gradients use the same vector for text data as the starting point of the integral).

However, Integrated Gradients satisfy two fundamental axioms: (a) Implementation Invariance, and (b) Sensitivity, making it a reliable and useful method for feature attribution. The process is defined across the i^{th} dimension for input x and baseline x' and creates a path between them by interpolating intermediate points. Computing the integral of their computed gradients gives the contribution to the output. Further down, it is shown the

approximation of the calculation:

$$\text{IntegratedGradients}_i(x) = (x_i - x_i') \times \sum_{k=1}^{m} \frac{\partial F(x' + \frac{k}{m} \times (x - x'))}{\partial x_i} \times \frac{1}{m} \tag{5.1}$$

Based on the development of Integrated Gradients, the authors in [24] introduced a variation of the algorithm with respect to the layers of the model called Conductance. The mathematical description is:

$$\text{Cond}_y^i(x) = (x_i - x_i') \cdot \int_0^1 \frac{\partial F(x' + a(x - x'))}{\partial y} \cdot \frac{\partial y}{\partial x_i} \partial a \tag{5.2}$$

$$\text{Cond}_y(x) = \sum_i (x_i - x_i') \cdot \int_0^1 \frac{\partial F(x' + a(x - x'))}{\partial y} \cdot \frac{\partial y}{\partial x_i} \partial a \tag{5.3}$$

Equation (1) defines the conductance of a neuron y for the attribution to an input variable i, and Equation (2) the total conductance of the hidden neuron y by summing over the input variables. We use the former to explain the function of the neuron in terms of its effect on the base features of the input and the latter to discuss the importance of the neuron. We can sum over the conductance of the neurons in a set of logically related neurons to define the conductance of the set as a whole.

The evaluation of the method is assessed by the metrics: 1. Activation: The value of the hidden unit is the feature importance score. 2. Internal Influence: The measure of feature importance is:

$$\text{IntInf}_y(x) ::= \int_0^1 \frac{\partial F(x' + \alpha \times (x - x'))}{\partial y} \partial \alpha \tag{5.4}$$

3. Gradient $\times$ Activation: $y \times \frac{\partial F(x' + \alpha \times (x - x'))}{\partial y}$.

5.2.2 External Explainers based on Model Behaviors

The most known framework for interpretability is SHAP [96]. As a fundamental principle, it hypothesizes that there are an original $f(x)$ and an explanation model $g(x')$ and $f(x) = g(x')$, with x' being a simplified input. It relies on three properties: (a) Local accuracy, (b) Missingness, and (c) Consistency.

The main goal is to find the features ϕ_i such that:

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x_i' \tag{5.5}$$

To achieve that goal, the process is simple. For each feature, you obtain all the possible x' inputs excluding it and get the result. However, due to the complexity of the process, calculating optimizations have been developed, with the most important being Kernel SHAP (Linear LIME [95] + Shapley values) and DeepSHAP (DeepLIFT + Shapley values).

First, we create a background dataset used for each feature not to be omitted during the process. Then we get the average of the model output for the specific feature. Finally,

it is converted into a linear regression problem with the coefficients being the bases ϕ_i.

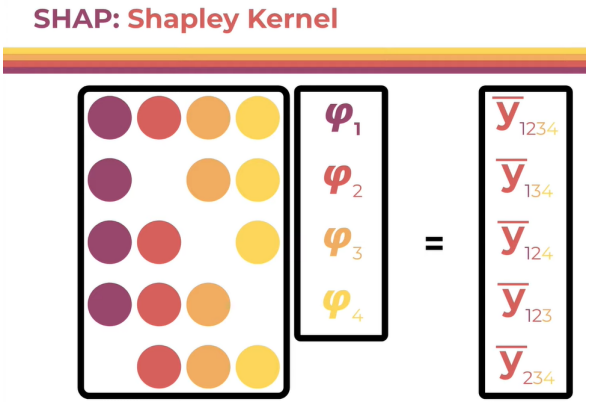


Figure 5.1. *Linear Regression for finding ϕ_i.*

As for DeepSHAP, it combines SHAP values computed for smaller components of the network into SHAP values for the whole network. It does so by recursively passing DeepLIFT's multipliers, now defined in terms of SHAP values, backward through the network.

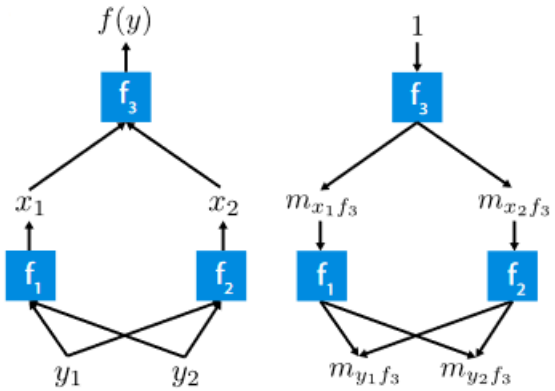


Figure 5.2. *Compositional models such as deep neural networks are comprised of many simple components. Given analytic solutions for the Shapley values of the components, fast approximations for the full model can be made using DeepLIFT's style of back-propagation.*

5.2.3 Contextual Decomposition

Murdoch et al. [97] present an interpretation algorithm for analyzing individual predictions made by LSTMs. By decomposing the output, the goal is to capture the contributions of combinations of words or variables to the final prediction.

$$o_t = \sigma(W_o x_t + V_o h_{t-1} + b_o) \tag{5.6}$$

$$f_t = \sigma(W_f x_t + V_f h_{t-1} + b_f) \tag{5.7}$$

$$i_t = \sigma(W_i x_t + V_i h_{t-1} + b_i) \tag{5.8}$$

$$g_t = \tanh(W_g x_t + V_g h_{t-1} + b_g) \tag{5.9}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{5.10}$$

$$h_t = o_t \odot \tanh(c_t) \tag{5.11}$$

Where W_o, W_i, W_f, $W_g \in \mathbb{R}^{d1 \times d2}$, V_o, V_f, V_i, $V_g \in \mathbb{R}^{d2 \times d2}$, b_o, b_g, b_i, $b_f \in \mathbb{R}^{d2}$ and $\odot$ denotes element-wise multiplication. o_t, f_t and i_t are often referred to as output, forget and input gates, respectively, due to the fact that their values are bounded between 0 and 1, and that they are used in element-wise multiplication.

The main idea (for LSTMs) starts by rewriting the equations of the model for h_t and c_t as sums of two factors:

$$h_t = \beta_t + \gamma_t$$

$$c_t = \beta_t^c + \gamma_t^c$$

The decomposition is constructed so that β_t corresponds to contributions made solely by the given phrase to h_t, and γ_t corresponds to contributions involving, at least in part, elements outside of the phrase.

Then they substitute the final equation:

$$p_j = \text{Softmax}((Wh^T)_j) = \frac{\exp(W_j h^T)}{\sum_{k=1}^{C} \exp(W_k h_t)}$$

by replacing the above relations:

$$p = \text{Softmax}(W\beta^T + W\gamma^T)$$

The authors test their algorithm on the task of sentiment analysis, using the SST (Stanford Sentiment Treebank) and Yelp Polarity datasets. They showed that CD is able to capture the composition of phrases of differing sentiment and extract instances of positive and negative negation.

The above algorithm was evolved in 2019 by Singh et al. [98], describing a new algorithm called Agglomerative Contextual Decomposition (ACD). Given a prediction from a trained DNN, ACD produces a hierarchical clustering of the input features, along with the contribution of each cluster to the final prediction. They use SST and ImageNet for their experiments.

ACD is a procedure for hierarchical clustering, where the CD interaction is used as the joining metric to determine which clusters to join at each step. The method has been modified to be applicable to all DNNs. For text models, an example is shown in Figure 4.

For hierarchical clustering, other algorithms have also been developed, such as SCD and SOC [101], which overcome the limitation of the previous two, as the β terms computed by both algorithms depend on the context of the phrase.

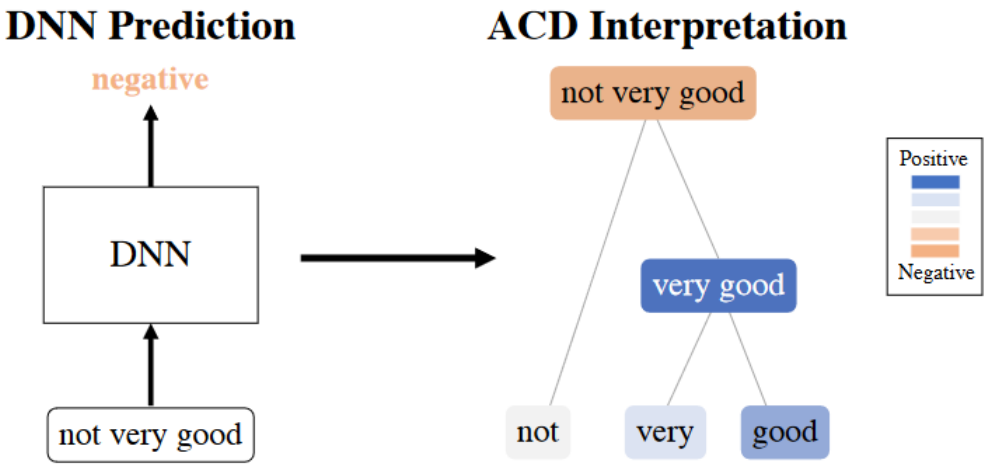


Figure 5.3. *ACD constructs a hierarchy of meaningful phrases and provides importance scores for each identified phrase.*

5.3 Applied Methods in LLMs

5.3.1 Discretized Integrated Gradients

In [23], the authors introduce a variant of Integrated Gradients (IG) [22] tailored for explaining the behavior of language models. They develop two interpolation strategies for the discrete word embedding space, generating points that lie close to actual words in the embedding space, resulting in more faithful gradient computation.

The key idea is to interpolate points that are closer to actual word embeddings, ensuring the gradients are meaningful. They propose a new interpolation algorithm and two search algorithms for the interpolated points: (a) Greedy, (b) MaxCount.

The method is tested using the following datasets:

1. SST2 dataset: 6920/872/1821 example sentences in the train/dev/test set, with the task being binary classification into positive/negative sentiment.

2. IMDB dataset: 25000/25000 example reviews in the train/test sets with binary labels for positive and negative sentiment.

3. Rotten Tomatoes (RT) dataset: 5331 positive and 5331 negative review sentences.

They used the processed datasets made available by the HuggingFace Dataset library and probed pre-trained BERT, DistilBERT, and RoBERTa text classification models individually fine-tuned for SST2, IMDB, and RT datasets.

The evaluation metrics used are:

1. Log-odds (LO) score: Average difference of the negative logarithmic probabilities on the predicted class before and after masking the top k% words with zero padding.

2. Comprehensiveness (Comp) score [102]: Measures the influence of the top-attributed words on the model's prediction.

3. Sufficiency (Suff) score [102]: Measures the adequacy of the top k% attributions for the model's prediction.

5.3.2 Sequential Integrated Gradients

The classic method of Integrated Gradients, as developed in [22], presents a significant limitation, shared by all path-based methods. These methods produce a path for each word of a sentence simultaneously, which can result in sentences with no clear meaning or significantly different meanings compared to the original.

Enguehard et al. [103] describe a new variation of Integrated Gradients (IG) called Sequential Integrated Gradients (SIG). The main idea is to compute the importance of each word in a sentence by keeping the other words fixed, creating interpolations only between the baseline and the word of interest. Additionally, they propose replacing the baseline token "pad" with the trained token "mask". Figure 5.4 illustrates the three variations of integrated gradients:

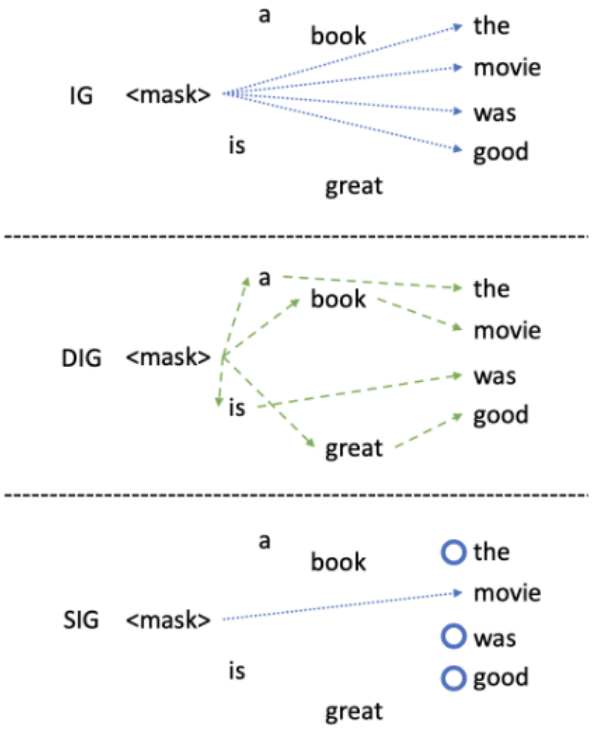


Figure 5.4. *Comparison between IG, DIG, and SIG. DIG improves on IG by creating discretized paths between the data and the baseline, but it can produce sentences with different meanings compared to the original. SIG addresses this by fixing every word to its true value except one, and moving the remaining word along a straight path.*

Denoting $F(x) : \mathbb{R}^{m \times n} \to \mathbb{R}$ as the language model, x_i as the i-th word of a sentence, and x_{ij} as the j-th feature of the i-th word, the only difference lies in the baseline defined for each word as $x_i = (x_1, ..., <\text{mask}>, ..., x_m)$.

The algorithm was tested using the SST2, IMDB, and RT datasets for sentiment classification. They used Log-Odds, Comprehensiveness, and Sufficiency as evaluation metrics. The results indicate that SIG is the most efficient among the variations. Contrary to the findings in [23], they found that IG using "mask" as a token outperforms DIG. The authors argue against the intuition in [23] that the discrete nature of the embedding space is crucial for explaining a language model.

However, SIG has some limitations, which are also proposed as future work. Firstly, SIG presents poor time complexity as it depends on the number of words in the input data. They noted that reducing the number of steps still yields better performance than IG with more steps. Therefore, for computing attributions on long sentences or texts, they recommend using SIG with a reduced number of steps instead of IG.

To alleviate the complexity, they implemented the possibility of computing gradients in parallel, using an internal batch size similar to how Captum [104] implemented IG. Secondly, they suggest validating the robustness of the method on more languages, tasks, and models in future work.

5.3.3 Layer Integrated Gradients for Linguistic Acceptability

In [105], the authors use Conductance or Layer Integrated Gradients (LIG) and Constituency Parse Trees (CPT) to explain the Linguistic Acceptability (LA) criteria learned by BERT on the Corpus of Linguistic Acceptability (CoLA) benchmark dataset [37]. They utilized the Captum library [106] for the LIG implementation and the Stanford CoreNLP toolkit for constructing the CPT.

The CoLA dataset includes tasks such as:

1. Causative-Inchoative Alternation (CIA)
2. Reflexive Antecedent Agreement (RAA)
3. Subject-Verb Agreement (SVA)
4. Subject-Verb-Object (SVO)
5. Wh-Extraction (WHE)

LIG is computed as the IG between the model output and a particular layer's input or output. The primary focus of their experiments relied on the LIG computed between the predicted class logit and the token embedding of the words. Further, they computed LIG heatmaps of CPT patterns w.r.t. the Input (Token + Segment + Position) embedding across the 12 Encoder layer embeddings of BERT.

The main idea is that computing the LIG for CPT patterns at different subtree levels can give insight into the constituents which contribute largely towards making the sentence LA or LUA. They conclude that the top subtree CPT patterns based on token embedding LIG were also dominant across the input and encoder layers of BERT. They observed that when the input strongly contributes towards a particular class (LA or LUA), the model has higher confidence in making the correct prediction. Additionally, a large percentage of

the misclassified sentences had negative LIG, indicating that the features disagreed with the model's prediction.

Based on these findings, they propose that future research could focus on improving the model's performance by parameterizing the LIG in the loss function during the later stages of the training process once the model has achieved reasonable performance. This could serve as a correction mechanism for the model.

5.3.4 Expected Gradients

In [107], Erion proposes an optimized version of Integrated Gradients called Expected Gradients. The new algorithm complements the main contribution of optimized attribution priors.

The main idea is to address the slow computation of IG when multiple references are involved. Choosing one specific reference is challenging because it implies that this specific example will not be highlighted as important. Expected Gradients avoid multiple integrals (as IG would require) by using sampling to yield accurate attributions with multiple references that can be calculated quickly.

Mathematically, Expected Gradients is defined as:

$$\text{ExpectedGradients}_i(x) := \int_{x'} ((x_i - x_i') \times \int_0^1 \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} d\alpha) p_D(x') dx' \tag{5.12}$$

where x is the target input, x' is the baseline input, and D is the underlying data distribution. Directly integrating over the training distribution is intractable; therefore, they reformulate the integrals as an expectation:

$$\text{ExpectedGradients}_i(x) := \mathbb{E}_{x' \sim D, \alpha \sim U(0,1)} \left[(x_i - x_i') \frac{\partial f(x' + \alpha(x - x'))}{\partial x_i} \right] \tag{5.13}$$

They tested the above method using attribution priors in image data, gene expression data, and health data where sparsity is desired.

5.3.5 Hierarchical Explanation

Chen et al. [108] propose a new divisive partitioning (DP) algorithm called HENCE, designed to generate hierarchical explanations by detecting feature interactions. Existing methods typically provide important features (words or phrases) selected from an input text as an explanation but often ignore the interactions between them.

The HENCE algorithm is outlined as follows:

Αλγοριθμος 5.1: *Hierarchical Explanation via Divisive Generation*

Input: text x with length n, and predicted label $\hat{y}$
Initialize the original partition $P_0 \leftarrow \{x_{(0,n]}\}$
Initialize the contribution set $C_0 = \emptyset$
Initialize the hierarchy $H = [P_0]$
for $t = 1$ to $n - 1$ **do**
    Find $x_{(s_i,s_{i+1}]}$ and j by solving Equation 1
    Update the partition $P'_t \leftarrow P_{t-1} \backslash \{x_{(s_i,s_{i+1}]}\}$
    $P_t \leftarrow P'_t \cup \{x_{(s_i,j]}, x_{(j,s_{i+1}]}\}$
    H.add(P_t)
    Update the contribution set C with $C'_t \leftarrow C_{t-1} \cup \{(x_{(s_i,j]}, \psi(x_{(s_i,j]}))\}$
    $C_t \leftarrow C'_t \cup \{(x_{(j,s_{i+1}]}, \psi(x_{(j,s_{i+1}]})\}$
end for
Output: C_{n-1}, H

Here, $P = \{x_{(0,s_1]}, x_{(s_1,s_2]}, \ldots, x_{(s_{P-1},n]}\}$ represents a partition of the word sequence with P text spans, where $x_{(s_i,s_{i+1}]} = (x_{s_i+1}, \ldots, x_{s_{i+1}})$. The algorithm is based on two fundamental equations: one for the interaction score that guides the partitioning, and another for measuring the contribution of each feature to the model prediction.

The method was tested on LSTM, CNN, and BERT models using the SST and IMDB datasets. The evaluation metrics employed include:

1. **AOPC:** Measures local fidelity by deleting or masking top-scored words and comparing the probability change on the predicted label.

2. **Log-Odds:** Evaluates the change in log-odds after feature removal.

3. **Cohesion-Score:** A new metric proposed by the authors to measure the synergy of words within a text span by shuffling the words and observing the probability change on the predicted label.

5.3.6 TransSHAP

Explanations of individual instances are often visualized using histograms. However, this approach is insufficient for text-based classifiers where inputs are sequential and structurally dependent. Kokalj et al. [109] introduce a variant of SHAP, called TransSHAP, applied to BERT for text classification, along with an improved method of visualizing explanations that better reflects the sequential nature of input.

The process of TransSHAP is depicted in Figure 5.5.

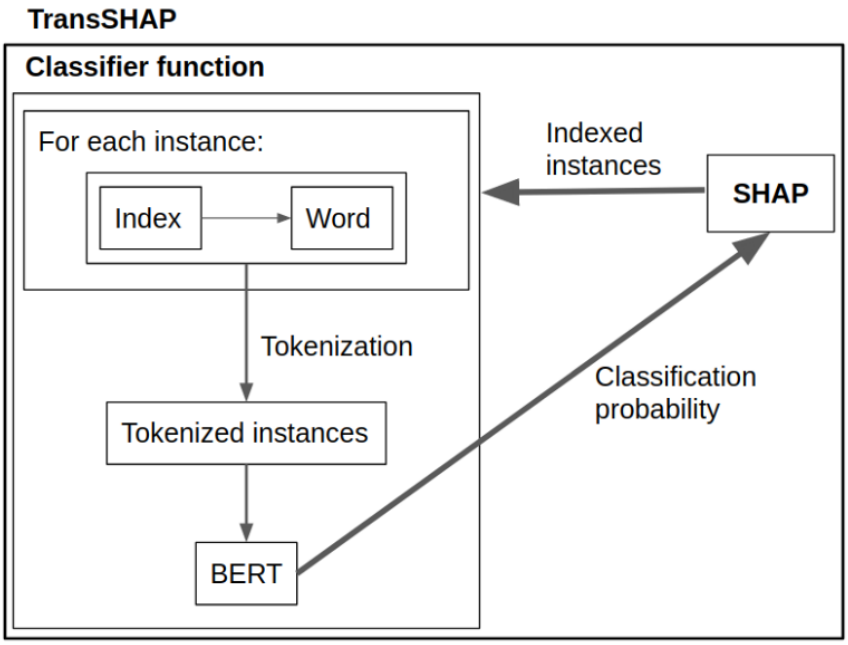


Figure 5.5. *TransSHAP adaptation of SHAP to the BERT language model by introducing a classifier function that converts each input instance into a word-level representation. The representation is perturbed to generate new instances, which are then processed by the BERT tokenizer, and the final predictions are returned to the Kernel SHAP.*

In this approach, the classifier function converts each input instance into a word-level representation. This representation is then perturbed to generate new instances, which are processed by the BERT tokenizer, and the final predictions are returned to the Kernel SHAP.

The method was tested on tweet sentiment classification using the CroSloEngual BERT model. The visualization strategy was enhanced by rotating the bar chart to a vertical position and presenting the features (words) in the order they appear in the original sentence.

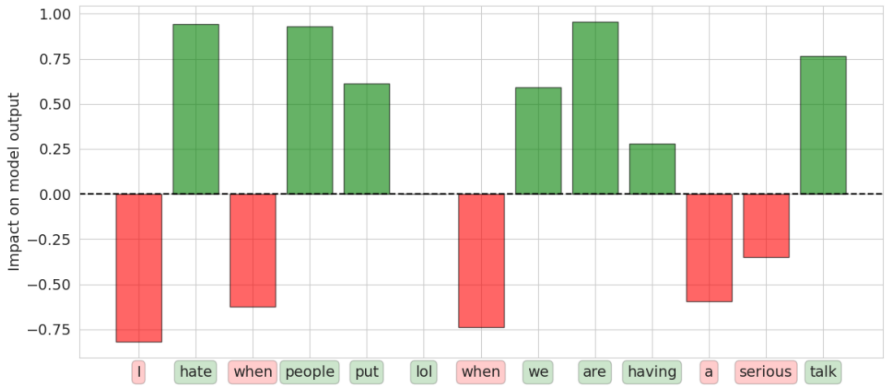


Figure 5.6. *TransSHAP visualization of prediction explanations for negative sentiment. The features' contribution values were obtained using the SHAP method. The word 'hate' strongly contributed to the negative sentiment classification, while the word 'lol' slightly opposed it.*

As future work, the authors suggest accounting for specific properties of text data and applying language models during the sampling step of the method. Additionally, they propose enhancing the explanations by expanding the features from individual words to larger textual units that are grammatically and semantically linked.

Part II

Methodology & Results

Chapter 6

Attribution Does Matter

6.1 Abstract

This research explores and optimizes attention mechanisms within Transformer models, a key component in Natural Language Processing (NLP) and various machine learning domains. The focus is on structured pruning of attention heads, a sophisticated technique aimed at optimizing models by selectively eliminating attention heads to preserve and enhance model performance while reducing computational complexity and resource requirements. This study introduces a novel approach that leverages the Neuron Conductance technique [16] for neuron attributions, examining the correlation between attention scores and their corresponding attributions. This approach provides a nuanced understanding of the interplay between attention mechanisms and the inherent importance of individual attention heads. A new metric, inspired by previous works, is developed to determine the importance of attention heads, facilitating more informed and effective pruning. Furthermore, this method is studied through the lens of the Lottery Ticket Hypothesis, demonstrating that our our metric competes with the approaches of Michel et al. [17] and Achlatis [18] giving results, which in the large datasets of GLUE are similar. Additionally, we conduct experiment for the method, Achlati's firstly proposed in his work, called "Iteratively Structured Pruning". The methodologies and implications of this research contribute not only a fresh perspective to model optimization techniques but also aim to advance the understanding of structured pruning and its impact on model performance, potentially leading to the development of more robust and resource-efficient models.

6.2 Introduction

Attention mechanisms, particularly those employed by Transformer models [4], have become foundational in NLP and various other domains within machine learning. These mechanisms enable models to weigh and prioritize different segments of the input sequence when generating an output, allowing for more nuanced and contextually aware representations of information. Within these mechanisms, attention heads are crucial for creating diverse linear transformations of input, enabling the model to focus on different aspects or features of the input data [7].

One of the most well-known Transformer-based models is BERT, proposed by Devlin

et al. [7]. 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, which slow down inference, increase the memory footprint, the number of computation operations (FLOPs), power usage, and contribute to environmental issues. This problem tends to rapidly scale up; for instance, BERT-base [7] contains 110 million parameters, while models like GPT-4[19], Google's Gemini[20], and Anthropic's Claude[21] contain hundreds of billions of parameters.

A response to this problem is model compression. Researchers have applied pruning techniques on BERT-base models by pruning weights or structured components such as attention heads. Researchers like Voita et al. [11], Kovaleva et al. [10], and Michel et al. [17] suggest that Transformer-based models are heavily overparameterized, allowing for the removal of a large number of heads without significant performance trade-offs.

This research aims to delve deeper into the methodologies and implications of attention head structured pruning, with a novel approach centered around a variant of Integrated Gradients [22] for obtaining attributions of the attention heads. Integrated Gradients have been a prominent method for feature importance in deep learning models, providing insights into model decisions [23]. This research leverages the Neuron Conductance technique [24], a method for neuron attributions concerning the model's output, with the following contributions:

- Examining the correlation matrix between attention scores and their corresponding attributions, offering a nuanced understanding of the interplay between attention mechanisms and the inherent importance of individual attention heads.

- Developing a novel metric based on the correlation between Neuron Conductance attributions and attention scores, facilitating more informed and effective pruning of less important heads.

- Drawing inspiration from previous works [17, 25], this research introduces an improved method for determining the importance of attention heads, enhancing the efficiency and efficacy of large language models.

This innovative approach builds on Achlatis' thesis [18], contributing a fresh perspective to the discourse on model optimization techniques. By integrating insights from the correlation between attention scores and attributions, this research aims to advance the field's understanding of structured pruning and its impact on model performance, paving the way for the development of more robust and resource-efficient models.

6.3 Related Work

Pruning in transformer models, particularly in architectures like BERT, has been extensively studied to enhance model efficiency without significant loss in performance. A foundational study by Michel et al. (2019) [17] demonstrated that many attention heads in transformers are redundant, challenging the belief that all heads are equally important. Their structured pruning approach iteratively removes the least important heads based on an importance metric calculated across the training set, revealing that the significance of attention heads is determined early in training and remains stable.

Building on this, Voita et al. (2019) [11] proposed a method for structured pruning that uses a heuristic function based on a Hard Concrete distribution to selectively retain only the most critical heads. Their findings align with Michel et al.'s [17], reinforcing the idea that a significant portion of attention heads can be pruned without degrading model performance.

The concept of pruning has also been explored from the perspective of the Lottery Ticket Hypothesis (LTH), introduced by Frankle and Carbin (2019) [9], which suggests that within a dense neural network lies a sparse subnetwork capable of matching the performance of the full model. While Frankle and Carbin focused on unstructured pruning in smaller networks, subsequent works like Liu et al. (2019) [26] and Chen et al. (2020) [13] tested LTH in larger models and more complex settings, including pre-trained BERT networks. Liu et al. (2019) [26] critiqued the generalizability of LTH, arguing that structured pruning in large models, especially under different optimization settings, may yield different outcomes than those observed in smaller, unstructured settings. On the other hand, Chen et al. (2020) [13] extended LTH to BERT, showing that subnetworks identified via magnitude pruning can perform comparably to the full model, particularly when initialized with pre-trained weights.

Further exploring the interplay between pruning and model interpretability, Prasanna et al. (2020) [12] and Yang et al. (2021) [27] investigated the effectiveness of structured pruning in understanding model behavior. Prasanna et al. [12] found that pruning based on importance metrics, particularly in BERT, could reveal insights into which heads and MLPs are most critical for task performance, with middle-layer heads often being more transferable across tasks. Yang et al. extended this by using attribution-based methods to guide pruning decisions, emphasizing the potential for task-specific compression in multi-task models.

Hao et al. (2021) [28]introduced a novel interpretability method called Self-Attention Attribution (AttAttr) that leverages attribution scores for pruning transformers. Their approach focuses on attention head connections, converting attribution scores into a pruning metric by taking the maximum attribution per connection and averaging it across all samples. This method shares conceptual similarities with Integrated Gradients but is specifically designed to target attention heads. Unlike methods grounded in the Lottery Ticket Hypothesis, Hao's approach incrementally prunes attention heads by masking them, rather than permanently removing them, and evaluates the pruned models without re-initializing the weights. This technique highlights a critical difference from my approach,

where the correlation between attention scores and attribution scores is calculated to guide pruning, with attributions computed concerning the model's predictions rather than the gold labels as in Hao's work.

Recent advancements by Ilhan et al. (2024) [29] and Grover et al. (2023) [30] further refined pruning techniques by introducing resource-efficient methods and interpretability-based pruning, respectively. Ilhan et al. [29] proposed a heuristic-based pruning method that optimizes the fine-tuning process by selectively updating weights, offering a practical solution for large-scale model deployment. Meanwhile, Grover et al. [30] addressed the challenge of noisy gradients in pruning, introducing methods that incorporate gradient information directly into the pruning metric, demonstrating superior performance in deep pruning scenarios.

Collectively, these studies illustrate the evolving understanding of pruning in transformer models, with structured and unstructured methods offering complementary insights. From identifying redundant components within BERT to leveraging pruning for interpretability and efficiency, the research highlights the critical balance between model complexity and performance, guiding the development of more efficient NLP models.

6.4 Problem Definition

6.4.1 Structured Pruning Definition

Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{n}$ and a desired sparsity level κ, neural network structured pruning can be formulated as the following constrained optimization problem:

$$\begin{aligned} \min_{w_s} L(w_s; D) &= \min_{w_s} \frac{1}{n} \sum_{i=1}^{n} \ell(w_s; (x_i, y_i)) \\ \text{s.t.} \quad w_s \in \mathbb{R}^m, &\quad \|w_s\|_0 \leq \kappa \end{aligned} \tag{6.1}$$

Here, $\ell(\cdot)$ represents the standard loss function, 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 $\|\cdot\|_0$ denotes the L0 norm.

Minimizing the L0 norm is challenging because it is non-convex, NP-hard, and requires combinatorial search, making structured pruning an NP-hard problem. Structured pruning can be performed before, during, or after fine-tuning. In this work, we first fine-tune the model and then apply our pruning methodology.

6.4.2 BERT Architecture

BERT consists of a stack of Transformer encoder layers [4]. Each layer has an identical structure: a multi-head self-attention (MHAtt) block followed by a multi-layer perceptron (MLP), with residual connections around each.

The MHAtt block contains N_h independently parameterized heads. An attention head h in layer l is parameterized by matrices $W_k^h, W_q^h, W_v^h \in \mathbb{R}^{d_h \times d}$ and $W_o^h \in \mathbb{R}^{d \times d_h}$, where

d_h is typically set to d/N_h. Given n d-dimensional input vectors $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^d$, MHAtt is computed as the sum of the outputs of each individual head applied to the input x:

$$\text{MHAtt}(x, q) = \sum_{h=1}^{N_h} \text{Att}(W_k^h, W_q^h, W_v^h, W_o^h)(x, q) \tag{6.2}$$

To allow different attention heads to interact with each other, Transformers apply a non-linear feed-forward network over the MHAtt output at each layer of the Transformer [4]. Each attention head may focus on different aspects of the input, such as syntax and semantics.

6.5 Proposed Method

6.5.1 Method Description

Firstly, as Michel et al. [17] 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 Multi-Head Attention (MHAtt):

$$\text{MHAtt}(x, q) = \sum_{h=1}^{N_h} \xi_h \cdot \text{Att}_{W_k^h, W_q^h, W_v^h, W_o^h}(x, q) \tag{6.3}$$

This modification helps us to mask the pruned heads without actually pruning the heads.

We define a new metric, based on the correlation between the attention scores of the tokens and the corresponding attributions:

$$I_{L,H} = \mathbb{E}_x \left(\text{corr} \left(\text{Attr}(A_{L,H}), A_{L,H} \right) \right) \tag{6.4}$$

where x is the data distribution, $A_{L,H}$ the attention score of the L layer and H head, and $\text{Attr}(A_{L,H})$ the attribution of this attention head. Both A and $\text{Attr}(A)$ have the same dimension of `num_tokens` $\times$ `num_tokens`.

For the attribution, we are using Neuron Conductance, which determines if the attention connection (i, j) has a significant influence on the model's prediction. Inspired by Hao et al. [25], we use correlation instead of the max function for aggregating the information.

The goal here is to treat each attention head as a variable x_i and each attribution as a variable y_i, with the measurements of each variable for computing the correlation being the connections of the token relations in each array.

In our research, we propose a new metric, that is used for Structured Pruning. Our contribution to this field is based on 2 central algorithms. These are the main ideas

Achlati's mentioned in his work [18].

Αλγοριθμος 6.1: *Structured Pruning with Importance Score*

Input: Fine-tuned pre-trained network
Set the initial pruning mask of attention heads to $s = \mathbf{1}_d$ ▷ where d is the dimension
repeat
    Calculate I_h for the non-pruned attention heads
    Sort heads in descending order based on I_h
    Prune $\kappa\%$ of initial heads with the lowest I_h and update s
until the sparsity of s reaches s_T ▷ s_T: Sparsity Threshold
Output: Pruning mask s

Αλγοριθμος 6.2: *Iterative Structured Pruning (ISP)*

Input: Fine-tuned pre-trained network
Set the initial pruning mask of attention heads to $s = \mathbf{1}_d$
repeat
    Calculate I_h for the non-pruned attention heads
    Sort heads in descending order based on I_h
    Prune $\kappa\%$ of remaining heads with the lowest I_h from the pre-trained model and update s
    Fine-tune the pre-trained network
until the sparsity of s reaches s_T ▷ s_T: Sparsity Threshold
Output: Pruning mask s

As for Algorithm 6.1, we use this as Achlati's proposed in his thesis. Firstly, we find the masks and evaluate in the dev set and then we extend the approach and apply the masks in the pre-trained model and fine tune it, in order to find winning tickets(Lottery Ticket Hypothesis). The last one method has been implemented also by Prasanna et al. [12]. We call it Integrated Gradients Structured Pruning Fine-Tuned (IGSPF).

In this point, inspiring by Michel's experiment implementation, we apply the masks on the pre-trained model in each iteration and fine tune it. This is similar, to the ISP Algorithm 6.2, but in this case in each iteration we prune a certain number of heads from the initial. We call it Integrated Gradients Structured Pruning Pre-trained (IGSPP).

As for the ISP Algorithm 6.2, we apply our metric to Achlati's idea to extend the IMP algorithm Chen proposed in his work [13], for Structured Pruning. The main difference here from the first algorithm, is that in each iteration, the prune ratio is constant, but number of heads is not. This is based on the LTH that the Iterative Algorithms could find more compact and easily learnable sub-networks.

6.5.2 The Concept Behind Correlation

Essentially, the purpose is to maintain heads with a larger positive monotonic relationship between attention scores and attributions. That is, the more general behavior of the attention head is that when an attention connection increases, its contribution to

prediction increases at the same time.

The motivation

In my research, I focused on the practical application of structured pruning in NLP models, using a novel metric that correlates attention mechanisms with attributions derived from the Neuron Conductance method—a refined version of Integrated Gradients. This work was motivated by the ongoing debate about whether attention mechanisms can serve as reliable explanations for model behavior, as highlighted in the papers "Attention is Not Explanation" [31] and "Attention is Not Not Explanation." [32].

The key finding from my research is that by ensuring attention weights align with robust attribution methods, we can preserve important interpretive features of a model even after substantial pruning. This not only challenges the notion that attention mechanisms are unreliable as explanations but also offers a practical framework for model optimization that maintains a high degree of interpretability. My work contributes to the debate by demonstrating that attention, when thoughtfully integrated with attribution methods, can be a valuable tool in both understanding and improving NLP models.

The idea here is that the attentions on their own are not enough to provide useful information, so we combine stable attributions to see what happens, in order to contribute in both directions: Structured Pruning and Attentions as an Explanation.

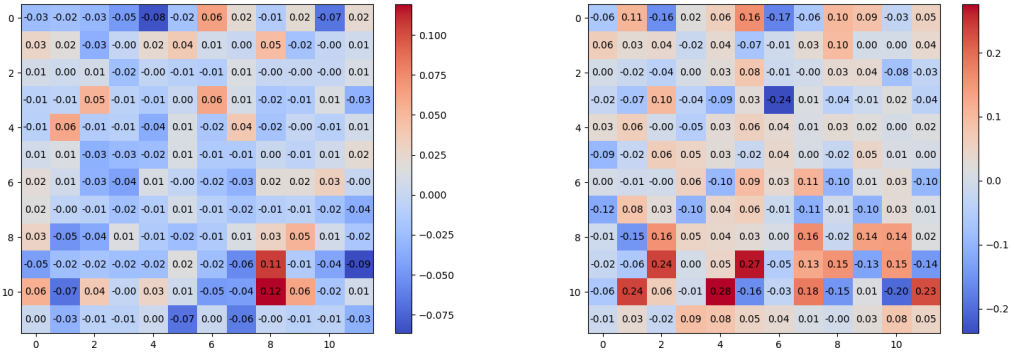
Spearman. Why?

The goal here is to keep the attention heads whose both variables have similar behavior, i.e., similar strength and direction between the connections.

Therefore, I chose Spearman's Rank Correlation over Kendall's Tau and Pearson's correlation due to its suitability for non-linear relationships and its sensitivity to rank differences. Pearson's correlation assumes linearity and is sensitive to outliers, making it less appropriate for attention and attribution scores, which may not follow a linear pattern. Spearman's Rank Correlation, however, captures monotonic relationships and provides a comprehensive evaluation of overall ranking agreement, making it more effective for assessing global trends. Additionally, Spearman's is more sensitive to significant rank discrepancies, offering a more nuanced understanding of the data compared to Kendall's Tau (used by Jain et al. [31]). This makes Spearman's the most suitable choice for analyzing the alignment between attention and attribution scores in this study.

Correlation Analysis

Firstly, we leverage a small subset of the IMDB, RTT, and SST-2 datasets, limiting our inputs to at most 200 tokens due to resource constraints. This subset is used to conduct a preliminary analysis of correlation values in both the pre-trained model (trained on the MLM task) and the fine-tuned model tailored to the corresponding tasks. In Figure 6.1, we present the correlation matrices for both the pre-trained and fine-tuned models, where the gradients have been computed with respect to the final prediction for SST-2. (See Appendix .1 for more datasets)



(a) Pre-trained Model *(b) Fine-tuned Model*

Figure 6.1. *Heatmaps for correlation of the pre-trained and fine-tuned model for the SST-2 dataset.*

As observed in Figure 6.1, there is a significant difference between the correlation values in the pre-trained and fine-tuned models, indicating that the attention weights have effectively incorporated task-specific knowledge during fine-tuning. Additionally, we observe the presence of attention heads with negative correlation values, which are strong candidates for pruning. Although the overall correlation values are not exceedingly high, there is still valuable information to be gleaned. Specifically, 3.47% of the attention heads for the IMDB dataset have correlation values between 0.2 and 0.4, while 2.08% and 4.86% of the heads in the RTT and SST-2 datasets, respectively, fall within this range. Notably, the SST-2 dataset yields more robust results in the pruning evaluation.

Metric Insights

Leveraging the fact that, there is valuable information in the models after fine tuning, we developed our metric (see Section 6.4) an in the following figures, we illustrate the behavior of our algorithm across various iterations. These visualizations display the attention scores alongside their corresponding attributions throughout the algorithm's progression, highlighting the selection criteria that guide the pruning process. It is important to note that the presented scores pertain to the pruned model, which reflects a decreasing number of heads as the iterations advance.

The visualizations focus on Layer 1 (index 0), since we have already find and apply the masks and fine-tuned the model. The illustration is only for one sample from MNLI dataset, thus it is a pair of sentences. As we transition from iteration 2 to iteration 3, Figures [6.2, 6.3], a distinct alteration is observed in the structure of the connections between attention scores and their corresponding attributions, particularly in attention head 1. Meanwhile, the other heads maintain a generally consistent structure.

Our primary concern lies in the connections between tokens, rather than merely the magnitude of the attention scores and attributions. This approach ensures that the overall structural integrity between the two components is preserved.

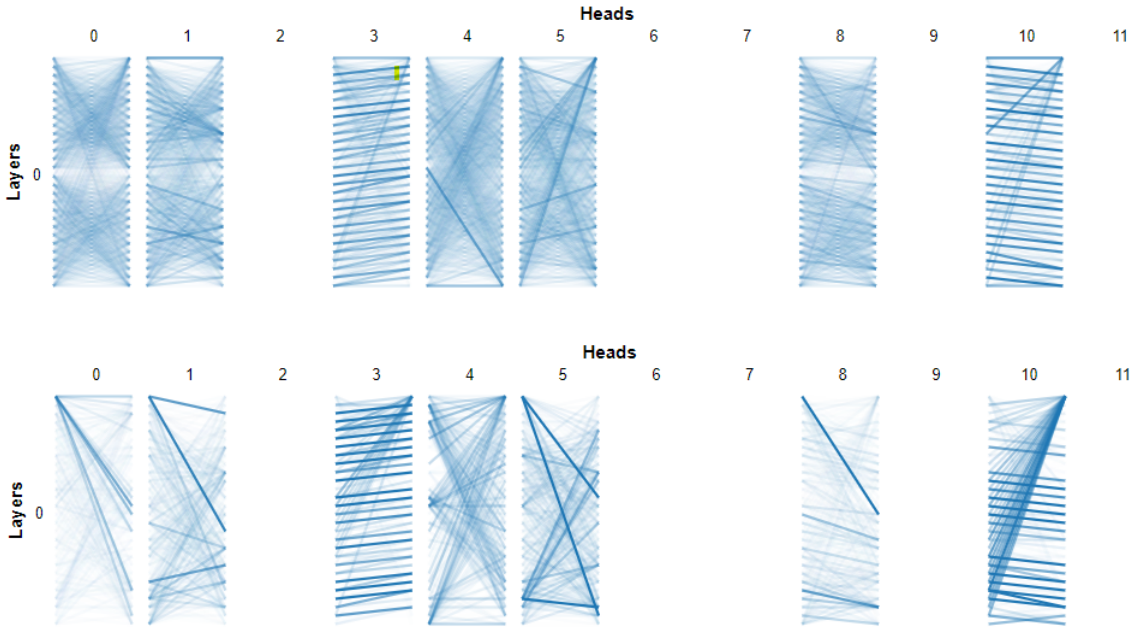


Figure 6.2. *Attention and Attribution Values on Iteration 2*

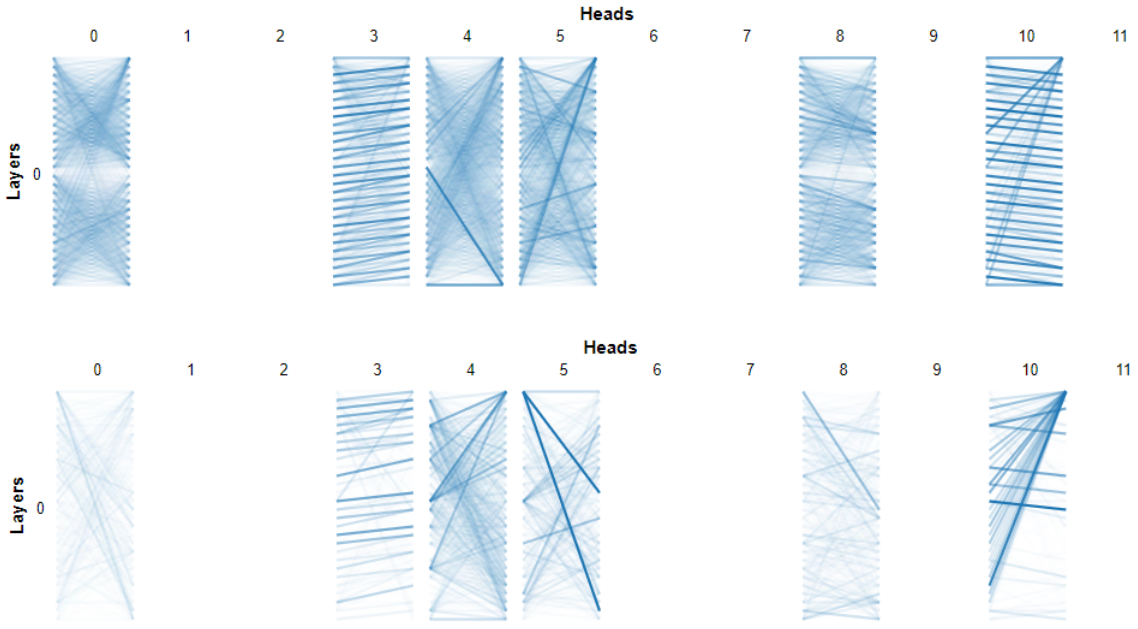


Figure 6.3. *Attention and Attribution Values on Iteration 3*

Finally, by the 6th iteration, Figure 6.4, we observe that the pruned model has reached a convergence point, where the structural similarities between attention scores and attributions are nearly identical across the remaining heads. This convergence signifies the algorithm's efficacy in selecting and preserving attention heads that contribute meaningfully to the model's interpretability and performance.

Figure 6.5 offers deeper insights into the correlation metric during iteration 2. Attention

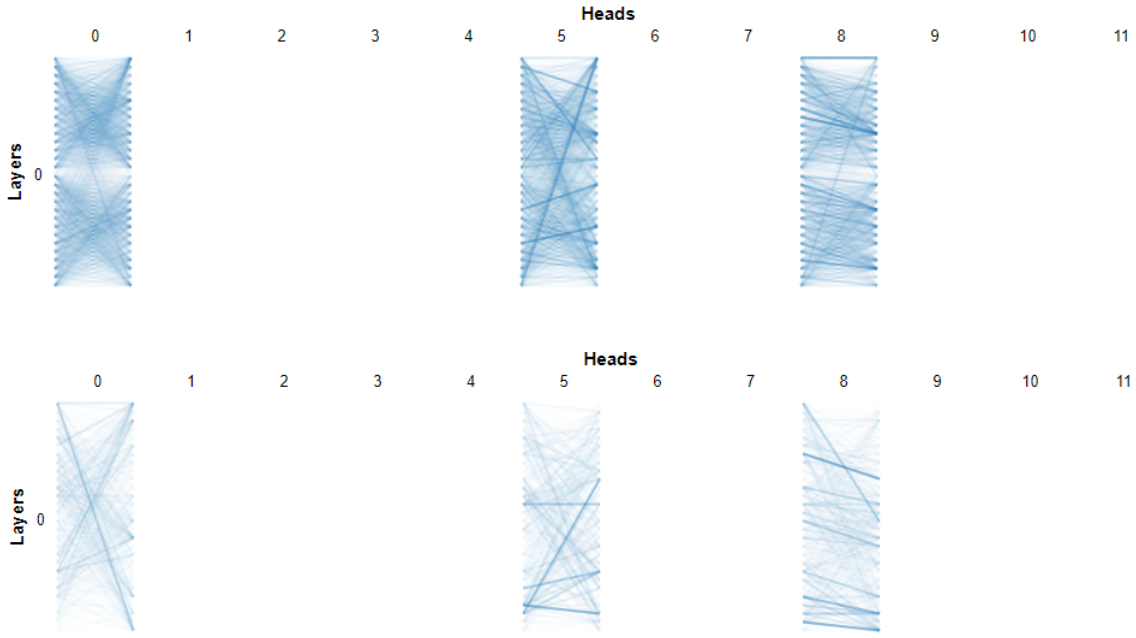
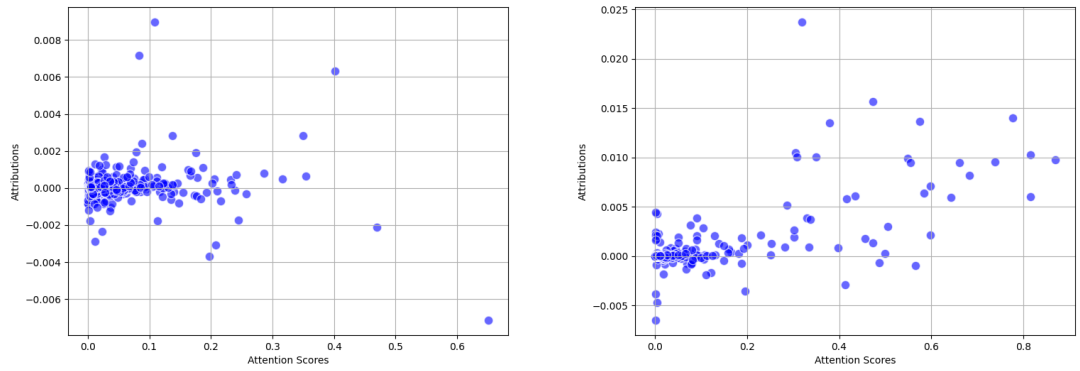


Figure 6.4. *Attention and Attribution Values on Iteration 6*



(a) Attention Head 1 *(b) Attention Head 3*

Figure 6.5. *Scatter Plots that shows the spatial relation of the heads 1, 3 on the iteration 2*

head 1 is a clear candidate for pruning, while head 3 is evidently worth retaining. This conclusion is apparent from the observed distribution of data points in the scatter plot.

6.6 Experiments

6.6.1 Configuration

For our experiments, we have used the classification tasks from GLUE Benchmark and compare the results with previous work.

- MNLI: Multi-Genre Natural Language Inference Corpus [33]
- QQP: Quora Question Pairs dataset
- QNLI: Question-answering NLI based on the Stanford Question Answering Dataset [34]
- MRPC: Microsoft Research Paraphrase Corpus [35]
- SST-2: Stanford Sentiment Treebank [36]
- CoLA: Corpus of Linguistic Acceptability [37]
- RTE: Recognizing Textual Entailment
- WNLI: Winograd Natural Language Inference Schema Challenge [38]

We have conducted experiments on the pre-trained BERT model, both "bert-base-uncased" 12-layers, 768-hidden, 12-heads, 110M parameters, (Transformers library) for various configurations.

Firstly, we have used a balanced subset of the the train set of each task to compute the metric. At the beginning, we run a statistic analysis in the datasets to find the region of the data distribution that the most information exist, and then we select randomly 2000 samples of these to compute the attributions and compute the metric. We used STD or Quantile method for the filtering. (see Appendix .2 for more details)

The evaluation is applied on the whole validation set for each task. The gradients have been calculated w.r.t. to the final prediction, i.e. after the model has already made the final decision.

We have conducted various experiments using our pruning metric. Firstly, we present in the following figures the behavior of each algorithm and then we present more compact our result, comparing them with Achlati's and Michel's work.

The pre-trained model is fine-tuned with the parameters in Table 6.1, being the same Achlatis used on his work [18].

Dataset	MNLI	QQP	QNLI	MRPC	SST-2	CoLA	RTE	WNLI
Train Examples	392,704	363,872	104,768	3,680	67,360	8,576	2,490	635
Iterations/Epoch	12,272	11,371	3,274	115	2,105	268	78	20
Epochs	3	3	3	3	3	3	3	3
Batch Size	32	32	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}	2×10^{-5}	2×10^{-5}
Optimizer	AdamW with $\epsilon = 1 \times 10^{-8}$							
Eval Metric	Matched Acc.	Accuracy	Accuracy	Accuracy	Accuracy	Matthew's	Accuracy	Accuracy

Table 6.1. *GLUE tasks [15], dataset sizes, metrics, and fine-tuning hyperparameters reported in this study.*

In addition to the algorithms i recently proposed, i have also conducted experiments to some others just for comparison.

1. **One Importance**: We calculate the importance score only once and then we apply various pruning ratios and evaluate on the dev set. This one was used in Michel's paper [17].

2. **One Shot ISP**: We conudct experiments on ISP for one iteration. Therefore, we calculate the first importance score and then we apply pruning ratio in the pre-trained model and then fine tune it. Someone, could tell, that this is the extension of the "One Importance" algorithm.

6.6.2 Structured Pruning with Importance Score

So, in the following Figures 6.6, we present our results for all the algorithms, put them together to the same plot.

As the results suggest, the IGSPF and IGSPP have literally better perforance than the baseline. However, the 2 baseline algorithms we use give pretty good results at low levels of pruning, if one considers the reduction of the time to find the subnetworks, as they do not need fine tuning.

In general, on large datasets, one may notice that our metric works quite well, since it keeps the perrformance quite high, even at pruning levels of 80%. While the two algorithms have essentially similar behavior, we note that IGSPP works better for smaller datasets, such as MRPC, WNLI, CoLA.

Noteworthy is the behavior of WNLI, where as the performance decreases on all datasets, during the iterations, WNLI, improves a lot. A possible interpretation is that BERT is overparameterized for such a small dataset which has a bad effect. By reducing the number of heads it seems that only the really useful ones remain. This is especially apparent in baseline algorithms, where they cut the heads during inference, which shows that too many heads during inference more divide the model than help it to make a decision. Also, we find that IGSPP, gives better results, which is reasonable, since it only finetunes the useful heads by metric from the beginning.

These differences in performances across the datasets, show the qualitative difference between the two algorithms IGSPF, IGSPP. In large datasets they do not differ much, and one could assume that the volume of the data, where the weights are combined in the final configuration, is responsible. Thus, the weights that have been deemed important in the end, whether you find them via fine tune or via prune in Pretrained and then finetune, will end up being the same groups. On small non-quality datasets, where weights have not converged, IGSPP, does relatively better as it continuously excludes weights from the original model and starts the process by changing the understanding of the weights that will be trained.

However, we also plot the *one shot ISP* (see Algorithm 6.2) in the same figure to see the improvement. It is obvious that it suprpasses the others and preserve high performance utill deep pruning rate such as 90%.

6.6.3 Iterative Structured Pruning

After that, we have conducted experiments for the Algorithm ISP (see Algorithm 6.2). The experiments that have been done concern only the datasets MNLI, QNLI as they require large computing resources and time and were tested only for one seed.

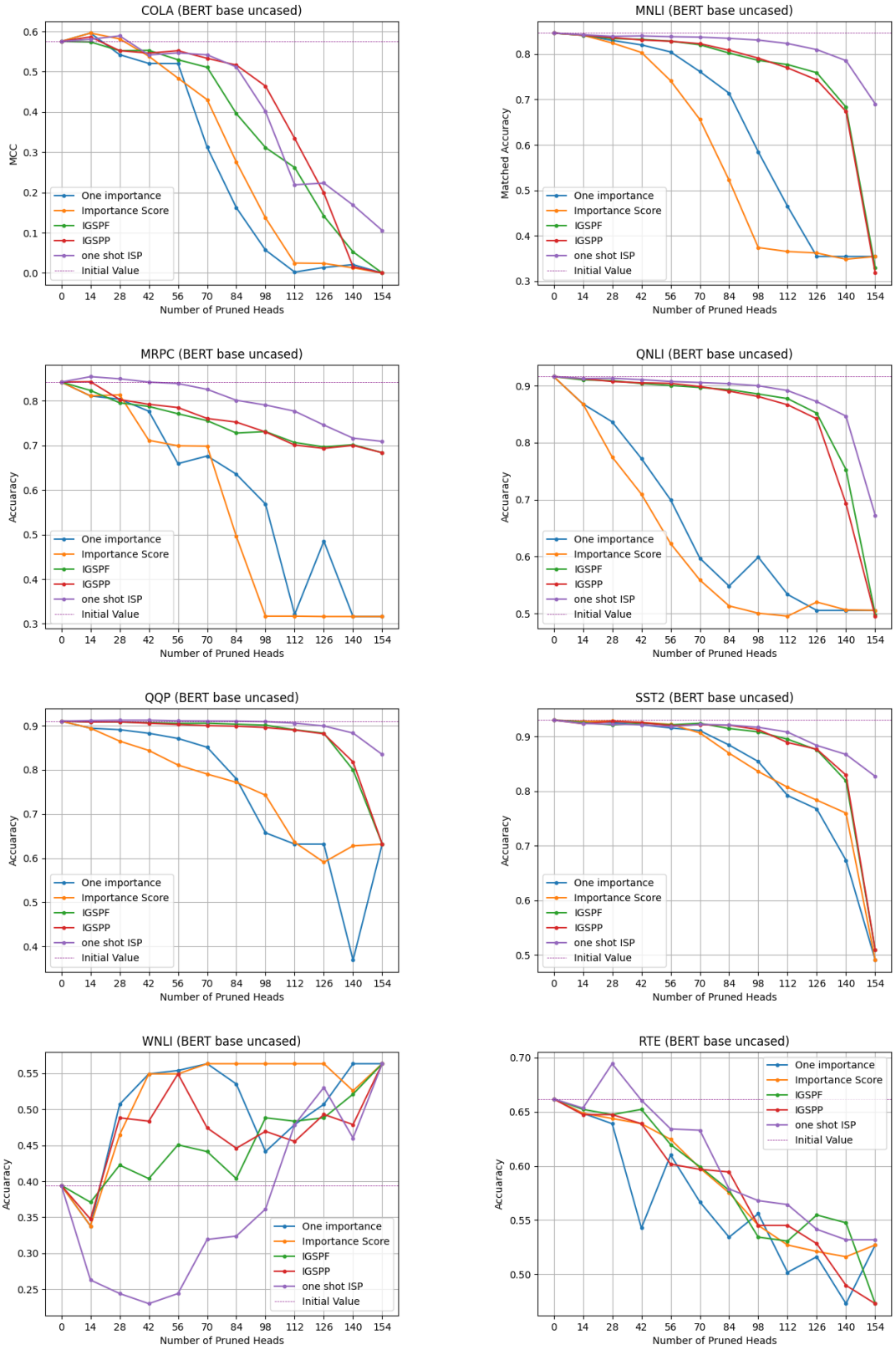


Figure 6.6. *Models evaluation on the validation set for each task for all the variations of the first Algorithm 6.1. The diagrams are the average value of three random seeds.*

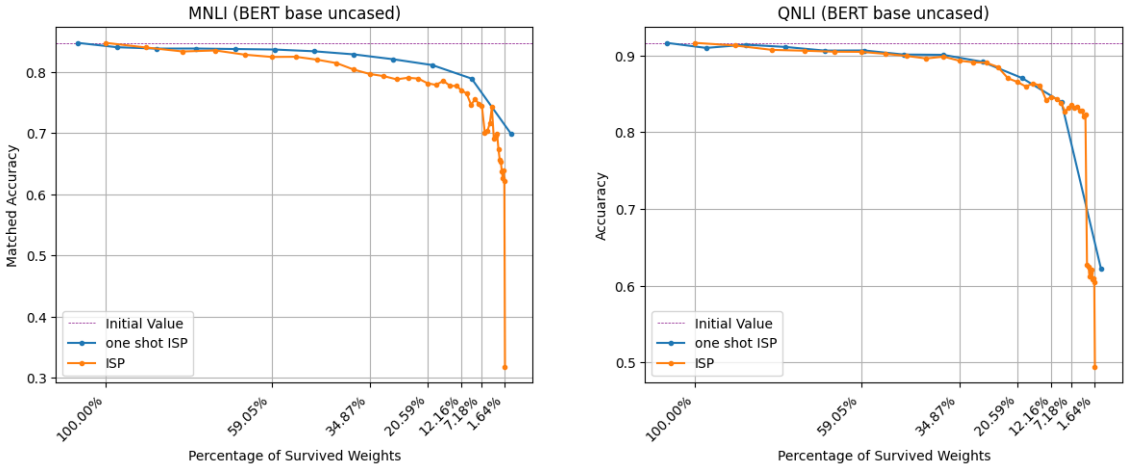


Figure 6.7. *Models evaluation on the validation set for ISP variations (see Algorithm 6.2). The experiments have been conducted for one seed.*

We observed that the ISP procedure in this configuration did not outperform the one-shot version. This suggests that the importance score obtained in the first iteration is highly effective, as it appears to capture a significant amount of crucial information. It's important to highlight that the one-shot ISP is essentially the "Once Importance" algorithm applied within the framework of the Lottery Ticket Hypothesis.

This finding is particularly valuable because it suggests that we might not need to run extensive, resource-intensive experiments to optimize the model, potentially saving significant time and computational resources.

6.6.4 Winning Tickets?

For some Algorithms, we are interested in if we could find the winning tickets by leveraging them. So, firstly it might be useful to define, based on Chen et al. [13] what is (i) a matching subnetwork, (ii) winning ticket and (iii) a universal subnetwork.

As he mentions in his paper:

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

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

$$\epsilon_T\left(A_T^t(f(x;m\odot\theta,\gamma))\right)\geq\epsilon_T\left(A_T^t(f(x;\theta_0,\gamma))\right)$$

Winning ticket. A subnetwork $f(x;m\odot\theta,\gamma)$ is a winning ticket for an algorithm A_T^t if it is a matching subnetwork for A_T^t and $\theta=\theta_0$.

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

Based on Chen, due to fluctuations in order to concider a subnetwork to be a winning ticket, it needs the performance of full BERT model to be within a standard deviation of the performance of the subnetwork(for the equality). Therefore, the condition, we use is that the full model performance should be less than the upper bound of the subnetwork performance.

Dataset	QNLI	QQP	MRPC	WNLI	RTE	SST-2	CoLA
Sparsity IGSPF				100%	30%		10%
Sparsity IGSPP			10%	100%	30%	30%	10%
Sparsity one shot ISP	20%	60%	30%	100%	30%		20%
Full BERT$_{\text{BASE}}$	91.5 ± 0.06	91.0 ± 0.07	84.2 ± 1.2	39.4 ± 0.8	66.1 ± 1.2	93.0 ± 0.3	57.5 ± 0.9
$f(x, m_{\text{IGSPF}} \odot \theta_0)$				56.3 ± 0.0	65.2 ± 2.2		57.3 ± 1.4
$f(x, m_{\text{IGSPP}} \odot \theta_0)$			84.3 ± 1.1	56.3 ± 0.0	63.8 ± 3.3	92.5 ± 0.4	58.6 ± 0.3
$f(x, m_{\text{one shot ISP}} \odot \theta_0)$	91.2 ± 0.2	91.0 ± 0.05	84.2 ± 0.3	56.3 ± 0.0	66.0 ± 1.6		58.8 ± 0.9

Table 6.2. *Winning Tickets across the datasets for IGSPF, IGSPP and one shot ISP*

It could be observed that a one-shot ISP (Iterative Sparsity Pruning) can identify winning tickets at high levels of sparsity in datasets where other algorithms fail to detect them entirely.

After that, we compare our results with Achlatis' best results, as a reference. The results have not been reproduced, so the comparison is approximate and qualitative.

At this point, we need to underline that the importance computation in Achlati's work, use the validation set, while we use balanced subset of the trainset.

To be aligned with Achlati's experiments, in Table 6.3, we present the results for pruning rate 70% and for the simple version without apply LTH, while in Table 6.4 there are results of LTH for pruning rate 80%.

Task	MNLI	QNLI	QQP	SST-2	MRPC	CoLA
Michel($\alpha = 1.0$)	0.717	0.734	0.791	0.875	0.639	0.286
Achlatis	**0.732**(0.4)	**0.787** (0.7)	**0.811** (0.6)	**0.878** (0.4)	**0.730** (0.6)	**0.387** (0.5)
Ours	0.374	0.500	0.742	0.836	0.317	0.137

Table 6.3. *Performance metrics for different tasks under methods, pruning and evaluating the fine tuned model. The greatest number is bold.*

Task	MNLI	QNLI	MRPC	SST-2	CoLA
Michel ($\alpha = 1.0$)	0.817	0.820	0.773	0.903	0.329
Achlatis	**0.824** (0.6)	**0.880** (0.4)	**0.783** (0.4)	**0.918** (0.4)	**0.435** (0.5)
(Ours) IGSPF	0.776	0.877	0.706	0.895	0.261
(Ours) IGSPP	0.769	0.866	0.701	0.889	0.334

Table 6.4. *Performance metrics for different tasks under methods, applying the Lottery Ticket Hypothesis. The greatest number is bold, while we underline the values that outperforms Michel's approach.*

Our metric in the simple algorithm could not surpass previous work and it is significantly less efficient. However, applying LTH, the results are better, outperforming Michel's approach for some tasks. In Table 6.4, we have also added the modified version of the Algorithm 6.1, using pre-trained model.

Chapter 7

Conclusions & Future Work

7.1 Discussion

This research has focused on the structured pruning of attention heads within Transformer models, particularly aiming to enhance model efficiency while maintaining interpretability and performance. A key contribution of this work is the development of a novel correlation-based metric between attention scores and Neuron Conductance attributions. This metric allowed for more effective pruning by identifying and preserving attention heads that exhibited strong positive relationships between these variables. This approach is designed to reduce the computational complexity of Transformer models, while still ensuring that the most critical components are retained.

To implement this, two structured pruning algorithms were introduced and evaluated. These algorithms demonstrated their ability to achieve significant model compression, particularly in smaller datasets, while maintaining high performance. The novel metric used in these pruning algorithms effectively identified attention heads that could be pruned without negatively impacting the model's overall effectiveness. The success of these methods illustrates that even highly parameterized models like BERT can be optimized without substantial loss of performance.

One of the most notable findings of this research is the performance of the one-shot ISP (Iterative Structured Pruning) method, which consistently provided the best results. The one-shot ISP method successfully identified "winning tickets" and maintained strong performance even at high levels of sparsity. This suggests that the method is particularly effective for structured pruning, positioning it as a preferred approach for model optimization when efficiency is critical.

Another significant contribution of this research is its relevance to the ongoing debate on whether "Attention is not Explanation." Our findings suggest that attention mechanisms, when combined with robust attribution methods like Neuron Conductance, can indeed provide meaningful explanations for model decisions. This highlights the value of attention not only as a computational tool but also as a mechanism that, when interpreted correctly, can offer insights into model behavior and decision-making processes. This contribution enriches the field of interpretability in machine learning, demonstrating that attention, when thoughtfully applied, remains a useful tool for model explanation.

7.2 Future Work

This research opens up several avenues for further exploration and improvement. The following directions are proposed for future work:

- **Attribution with Respect to the Golden Label:** An interesting extension of this work would involve calculating attributions with respect to the golden label instead of the model's prediction. This approach could provide deeper insights into how attention heads contribute to correct model predictions and could refine the pruning process further.
- **Testing Masks on Similar Tasks:** To assess the generalizability of the pruning masks developed in this study, future research could apply these masks to models fine-tuned on similar tasks. This would help determine whether the masks retain their effectiveness across different but related tasks, expanding their utility.
- **Transfer Learning from MLM Models:** Another promising direction is to explore transfer learning using masks derived from a Masked Language Model (MLM). Specifically, masks could be generated from the MLM model (targeting the index of the masked token) and then applied to fine-tuned models. This approach aligns with the concept of "universal tickets," suggesting that such masks could be widely applicable across different tasks.
- **Using Kendall Correlation:** Finally, to build on the findings of this research, Kendall's Tau correlation could be explored as an alternative to Spearman's Rank correlation. This is particularly relevant in light of the arguments made in the "Attention is not Explanation" paper, where Kendall's Tau was used to assess the reliability of attention as an interpretive tool. This comparison could offer new perspectives on the robustness of the pruning metric and potentially improve its effectiveness.

These future work directions aim to enhance the effectiveness, generalizability, and interpretability of structured pruning techniques, contributing to the development of more efficient and versatile models.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio και Aaron Courville. *Deep Learning*. MIT Press, 2016.

[2] Rafael C Gonzalez. *Digital image processing*. Pearson education india, 2009.

[3] Dzmitry Bahdanau, Kyunghyun Cho και Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. *arXiv e-prints*, σελίδα arXiv:1409.0473, 2014. _eprint: 1409.0473.

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

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

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

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

[8] 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*. *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, σελίδες 353–355, 2018.

[9] Jonathan Frankle και Michael Carbin. *The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks*. 2018. Publisher: arXiv Version Number: 5.

[10] Olga Kovaleva, Alexey Romanov, Anna Rogers και Anna Rumshisky. *Revealing the Dark Secrets of BERT*. *CoRR*, abs/1908.08593, 2019. arXiv: 1908.08593.

[11] 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*. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, σελίδες 5797–5808, Florence, Italy, 2019. Association for Computational Linguistics.

[12] Sai Prasanna, Anna Rogers και Anna Rumshisky. *When BERT Plays the Lottery, All Tickets Are Winning*. *Proceedings of the 2020 Conference on Empirical Methods*

in Natural Language Processing (EMNLP), σελίδες 3208–3229, Online, 2020. Association for Computational Linguistics.

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

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

[15] 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 preprint arXiv:1804.07461*, 2019.

[16] Kedar Dhamdhere. *How Important Is a Neuron?* 2018.

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

[18] Stefanos Achlatis. *Structured Pruning for Deep Learning Language Models*. PhD Thesis, National Technical University of Athens, 2021.

[19] OpenAI. *GPT-4 Technical Report*. 2023.

[20] Romal Thoppilan, Daniel M De Freitas, Jamie Hall, Noam Shazeer, Abhishek Kulshreshtha, Heng Tze Cheng και et al. *LaMDA: Language Models for Dialog Applications*. *arXiv preprint arXiv:2201.08239*, 2022.

[21] Yuntao Bai, Andy Jones, Kahin Ndousse, Amanda Askell, Anna Chen, Nisanth DasSarma και et al. *Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback*. *arXiv preprint arXiv:2204.05862*, 2022.

[22] Mukund Sundararajan, Ankur Taly και Qiqi Yan. *Axiomatic Attribution for Deep Networks*. *Proceedings of the 34th International Conference on Machine Learning*Doina Precup και Yee Whye Teh, επιμελητές, τόμος 70 στο *Proceedings of Machine Learning Research*, σελίδες 3319–3328. PMLR, 2017.

[23] Soumya Sanyal και Xiang Ren. *Discretized Integrated Gradients for Explaining Language Models*. *arXiv e-prints*, σελίδα arXiv:2108.13654, 2021. _eprint: 2108.13654.

[24] Kedar Dhamdhere, Mukund Sundararajan και Qiqi Yan. *How Important Is a Neuron?* *CoRR*, abs/1805.12233, 2018. arXiv: 1805.12233.

[25] Yaru Hao, Li Dong, Furu Wei και Ke Xu. *Self-Attention Attribution: Interpreting Information Interactions Inside Transformer*. *arXiv e-prints*, σελίδα arXiv:2004.11207, 2020. _eprint: 2004.11207.

[26] Zhuang Liu, Mingjie Sun, Zhijie Zhou, Gao Huang και Trevor Darrell. *Rethinking the Value of Network Pruning*. *International Conference on Learning Representations (ICLR)*, 2019.

[27] Kevin Yang, Tim Michaels, Yonatan Belinkov και James Glass. *Task-specific Compression for Multi-task Language Models using Attribution-based Pruning*. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, σελίδες 1828–1840, 2021.

[28] Shiyue Hao, Zhongyu Wei, Yiming Liu και Xiaoyu Su. *Self-Attention Attribution: Interpreting Information Interactions Inside Transformer*. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, σελίδες 4354–4364, 2021.

[29] Eren Ilhan, Arda Şenocak και Trevor Darrell. *Resource-Efficient Transformer Pruning for Finetuning of Large Models*. *arXiv preprint arXiv:2401.00501*, 2024.

[30] Vibhor Grover, Akshay Rathi, Nachiket Kapre και Shikhar Verma. *DeepCuts: Single-Shot Interpretability based Pruning for BERT*. *Proceedings of the 2023 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, σελίδες 1234–1245, 2023.

[31] Sarthak Jain και Byron C. Wallace. *Attention is not Explanation*. *CoRR*, abs/1902.10186, 2019. arXiv: 1902.10186.

[32] Sarah Wiegreffe και Yuval Pinter. *Attention is not not Explanation*. *CoRR*, abs/1908.04626, 2019. arXiv: 1908.04626.

[33] 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*, σελίδες 1112–1122, 2018.

[34] 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, 2016.

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

[36] 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, 2013.

[37] Alex Warstadt, Amanpreet Singh και Samuel R. Bowman. *Neural Network Acceptability Judgments*. *arXiv e-prints*, σελίδα arXiv:1805.12471, 2018. _eprint: 1805.12471.

[38] Ernest Davis, Hector J. Levesque και Leora Morgenstern. *The Winograd Schema Challenge*. *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*, 2011.

[39] Stuart J Russell και Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[40] Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013.

[41] Finale Doshi-Velez και Been Kim. *Towards A Rigorous Science of Interpretable Machine Learning*. *arXiv e-prints*, σελίδα arXiv:1702.08608, 2017. _eprint: 1702.08608.

[42] Yann LeCun, John Denker και Sara Solla. *Optimal Brain Damage*. *Advances in Neural Information Processing Systems*D. Touretzky, επιμελητής, τόμος 2. Morgan-Kaufmann, 1989.

[43] Emma Strubell, Ananya Ganesh και Andrew McCallum. *Energy and Policy Considerations for Deep Learning in NLP*. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, σελίδες 3645–3650, 2019.

[44] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lucy M Munguia, Daniel Rothchild και Jeff Dean. *Carbon Emissions and Large Neural Network Training*. *arXiv preprint arXiv:2104.10350*, 2021.

[45] Peter Henderson, Jieru Hu, David Romero, Gregory Larson, David Cafaro, W Clarke και Joelle Pineau. *Towards the Systematic Reporting of the Energy and Carbon Footprints of Machine Learning*. *Journal of Machine Learning Research*, 21(1):1–43, 2020.

[46] Kevin Clark, Urvashi Khandelwal, Omer Levy και Christopher D Manning. *What Does BERT Look at? An Analysis of BERT's Attention*. *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, σελίδες 276–286, 2019.

[47] Yang Liu και Mirella Lapata. *Text Summarization with Pretrained Encoders*. *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 3730–3740, 2019.

[48] Ian Tenney, Dipanjan Das και Ellie Pavlick. *BERT Rediscovers the Classical NLP Pipeline*. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, σελίδες 4593–4601, Florence, Italy, 2019. Association for Computational Linguistics.

[49] Mitchell A Gordon, Kevin Duh και Nicholas Andrews. *Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning*. *arXiv preprint arXiv:2002.08307*, 2020.

[50] Yoav Goldberg. *Assessing BERT's Syntactic Abilities*. *arXiv preprint arXiv:1901.05287*, 2019.

[51] Anna Rogers, Olga Kovaleva και Anna Rumshisky. *A Primer in BERTology: What We Know About How BERT Works*. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020. Place: Cambridge, MA Publisher: MIT Press.

[52] Victor Sanh, Lysandre Debut, Julien Chaumond και Thomas Wolf. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. *arXiv preprint arXiv:1910.01108*, 2019.

[53] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden και Alexandru Peste. *Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks*. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

[54] Naresh R Zadeh και Andreas Moshovos. *Gist: Efficient data encoding for deep neural network training*. *Proceedings of the 47th Annual International Symposium on Computer Architecture*, σελίδες 776–789, 2020.

[55] Trevor Gale, Erich Elsen και Sara Hooker. *The State of Sparsity in Deep Neural Networks*. *arXiv preprint arXiv:1902.09574*, 2020.

[56] Tom M Mitchell και Tom M Mitchell. *Machine learning*, τόμος 1. McGraw-hill New York, 1997.

[57] Christopher M Bishop και Nasser M Nasrabadi. *Pattern recognition and machine learning*, τόμος 4. Springer, 2006.

[58] Yijun Yuan, Jiawei Hou, Andreas Nüchter και Sören Schwertfeger. *Self-supervised Point Set Local Descriptors for Point Cloud Registration*. *CoRR*, abs/2003.05199, 2020. arXiv: 2003.05199.

[59] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong και Qing He. *A Comprehensive Survey on Transfer Learning*. *CoRR*, abs/1911.02685, 2019. arXiv: 1911.02685.

[60] Richard O Duda. *Peter E. hart, and David G. Stork, Pattern Classification, A Willey-Interscience*, 2001.

[61] Léon Bottou. *Large-Scale Machine Learning with Stochastic Gradient Descent*. *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*Yves Lechevallier και Gilbert Saporta, επιμελητές, σελίδες 177–187, Paris, France, 2010. Springer.

[62] Yuanzhi Li, Colin Wei και Tengyu Ma. *Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training Neural Networks*. *CoRR*, abs/1907.04595, 2019. arXiv: 1907.04595.

[63] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia και Kaiming He. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. *CoRR*, abs/1706.02677, 2017. arXiv: 1706.02677.

[64] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt και B. Scholkopf. *Support vector machines*. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.

[65] Sepp Hochreiter και Jürgen Schmidhuber. *Long short-term memory*. *Neural computation*, 9(8):1735–1780, 1997. Publisher: MIT press.

[66] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel και Yoshua Bengio. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. *CoRR*, abs/1502.03044, 2015. arXiv: 1502.03044.

[67] Kaiming He, Xiangyu Zhang, Shaoqing Ren και Jian Sun. *Deep Residual Learning for Image Recognition*. *CoRR*, abs/1512.03385, 2015. arXiv: 1512.03385.

[68] Jimmy Lei Ba, Jamie Ryan Kiros και Geoffrey E. Hinton. *Layer Normalization*, 2016. _eprint: 1607.06450.

[69] 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.

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

[71] 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.

[72] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee και Luke Zettlemoyer. *Deep Contextualized Word Representations*, 2018.

[73] 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.

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

[75] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer και V. Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. *arXiv preprint arXiv:1907.11692*, 2019.

[76] Kevin Clark, Minh Thang Luong, Quoc V. Le και Christopher D. Manning. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. *arXiv preprint arXiv:2003.10555*, 2020.

[77] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma και Radu Soricut. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. *arXiv preprint arXiv:1909.11942*, 2020.

[78] 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, 2017.

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

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

[81] 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.

[82] Joris Baan, Maartje ter Hoeve, Marlies van der Wees, Anne Schuth και Maarten de Rijke. *Understanding Multi-Head Attention in Abstractive Summarization*, 2019.

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

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

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

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

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

[88] 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.

[89] 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.

[90] 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.

[91] 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.

[92] 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, 2017. Place: Vancouver, Canada Publisher: Association for Computational Linguistics.

[93] Christos Louizos, Max Welling και Diederik P. Kingma. *Learning Sparse Neural Networks through L0 Regularization*. 2018.

[94] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh και Dhruv Batra. *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization*. *arXiv e-prints*, σελίδα arXiv:1610.02391, 2016. _eprint: 1610.02391.

[95] Marco Tulio Ribeiro, Sameer Singh και Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, σελίδες 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. event-place: San Francisco, California, USA.

[96] Scott Lundberg και Su In Lee. *A Unified Approach to Interpreting Model Predictions*. *arXiv e-prints*, σελίδα arXiv:1705.07874, 2017. _eprint: 1705.07874.

[97] W. James Murdoch, Peter J. Liu και Bin Yu. *Beyond Word Importance: Contextual Decomposition to Extract Interactions from LSTMs*. *arXiv e-prints*, σελίδα arXiv:1801.05453, 2018. _eprint: 1801.05453.

[98] Chandan Singh, W. James Murdoch και Bin Yu. *Hierarchical interpretations for neural network predictions*. *arXiv e-prints*, σελίδα arXiv:1806.05337, 2018. _eprint: 1806.05337.

[99] Avanti Shrikumar, Peyton Greenside και Anshul Kundaje. *Learning Important Features Through Propagating Activation Differences*. *Proceedings of the 34th International Conference on Machine Learning*Doina Precup και Yee Whye Teh, επιμελητές, τόμος 70 στο *Proceedings of Machine Learning Research*, σελίδες 3145–3153. PMLR, 2017.

[100] Pieter Jan Kindermans, Kristof Schütt, Klaus Robert Müller και Sven Dähne. *Investigating the influence of noise and distractors on the interpretation of neural networks*. *arXiv e-prints*, σελίδα arXiv:1611.07270, 2016. _eprint: 1611.07270.

[101] Xisen Jin, Zhongyu Wei, Junyi Du, Xiangyang Xue και Xiang Ren. *Towards Hierarchical Importance Attribution: Explaining Compositional Semantics for Neural Sequence Models*. *arXiv e-prints*, σελίδα arXiv:1911.06194, 2019. _eprint: 1911.06194.

[102] Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher και Byron C. Wallace. *ERASER: A Benchmark to Evaluate Rationalized NLP Models*. *arXiv e-prints*, σελίδα arXiv:1911.03429, 2019. _eprint: 1911.03429.

[103] Joseph Enguehard. *Sequential Integrated Gradients: a simple but effective method for explaining language models*. *arXiv e-prints*, σελίδα arXiv:2305.15853, 2023. _eprint: 2305.15853.

[104] Narine Kokhlikyan. *Captum: A unified and generic model interpretability library for PyTorch*.

[105] Anmol Nayak και Hari Prasad Timmapathini. *Using Integrated Gradients and Constituency Parse Trees to explain Linguistic Acceptability learnt by BERT*. *arXiv e-prints*, σελίδα arXiv:2106.07349, 2021. _eprint: 2106.07349.

[106] Narine Kokhlikyan. *Captum: A unified and generic model interpretability library for PyTorch*, 2020.

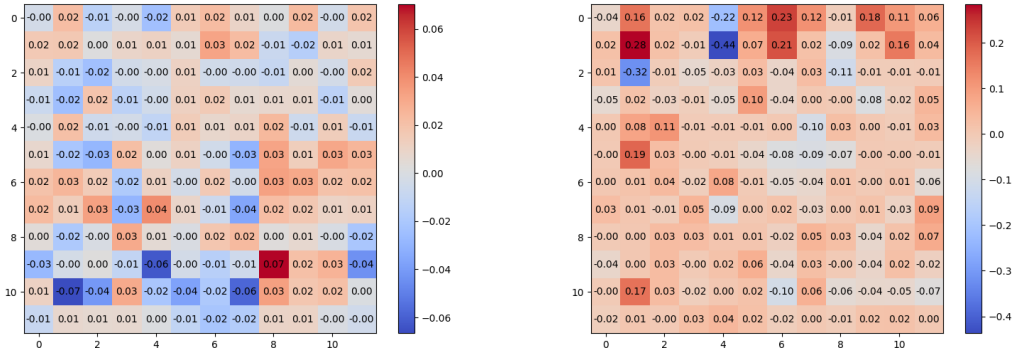
[107] Gabriel Erion, Joseph D. Janizek, Pascal Sturmfels, Scott Lundberg και Su In Lee. *Improving performance of deep learning models with axiomatic attribution priors and expected gradients*. *arXiv e-prints*, σελίδα arXiv:1906.10670, 2019. _eprint: 1906.10670.

[108] Hanjie Chen, Guangtao Zheng και Yangfeng Ji. *Generating Hierarchical Explanations on Text Classification via Feature Interaction Detection*. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, σελίδες 5578–5593, Online, 2020. Association for Computational Linguistics.

[109] Enja Kokalj, Blaž Škrlj, Nada Lavrač, Senja Pollak και Marko Robnik-Šikonja. *BERT meets Shapley: Extending SHAP Explanations to Transformer-based Classifiers*. *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*, σελίδες 16–21, Online, 2021. Association for Computational Linguistics.

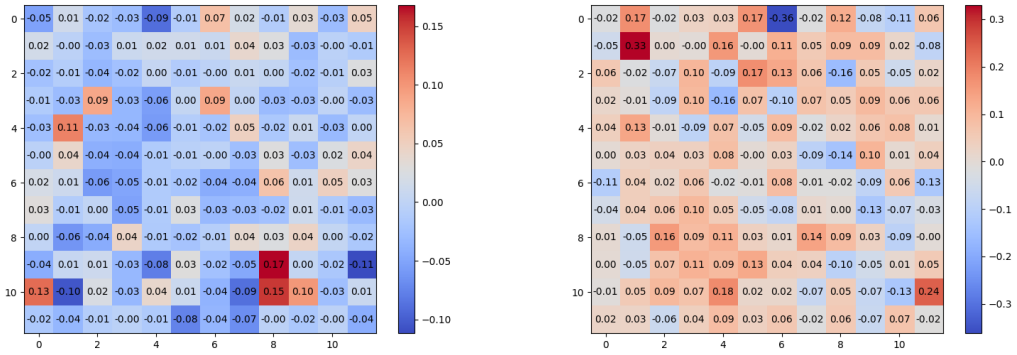
Appendices

.1 Correlation Analysis



(a) Pre-trained Model *(b) Fine-tuned Model*

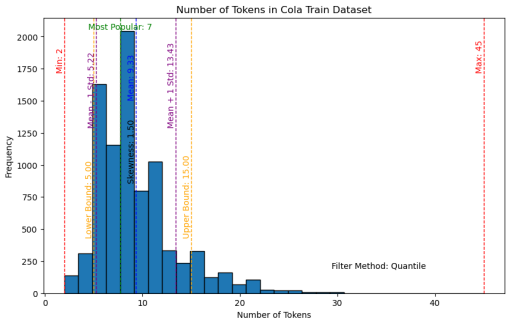
Figure 1. *Heatmaps for correlation of the pre-trained and finetuned model for the IMDB dataset*



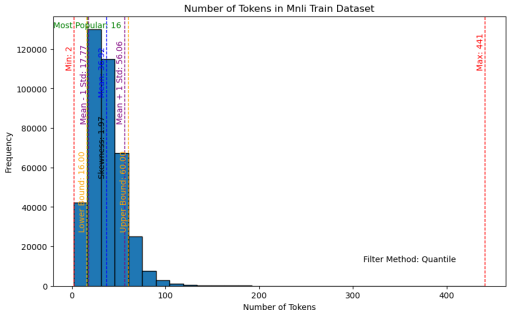
(a) Pre-trained Model *(b) Fine-tuned Model*

Figure 2. *Heatmaps for correlation of the pre-trained and finetuned model for the Rotten-Tomatoes dataset*

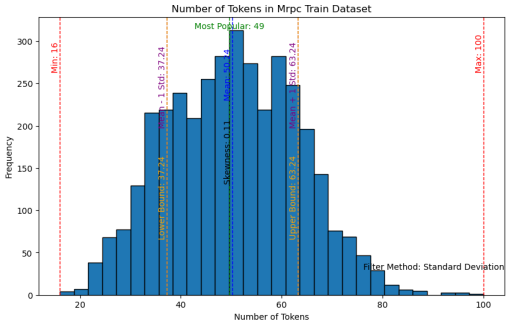
.2 Data Pre-Processing



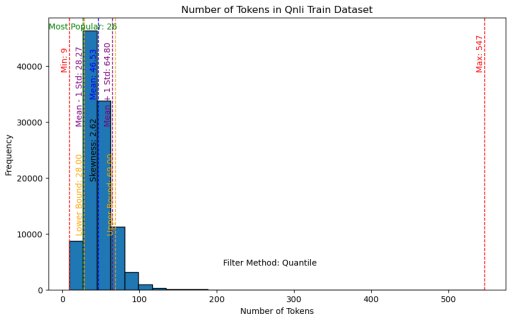
(a) CoLA Dataset



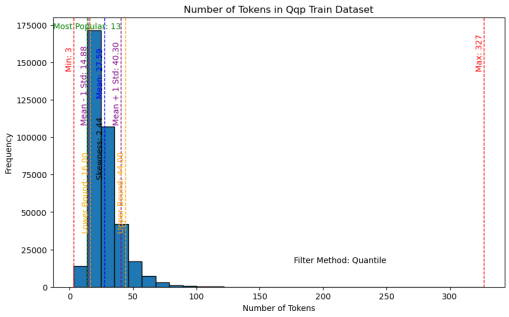
(b) MNLI Dataset



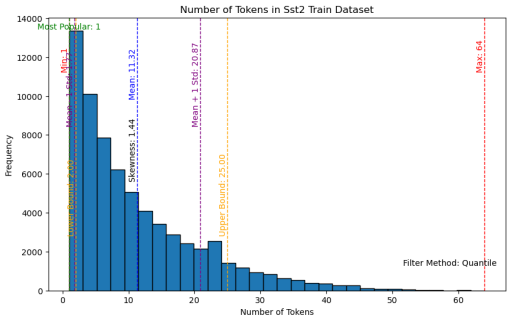
(c) MRPC Dataset



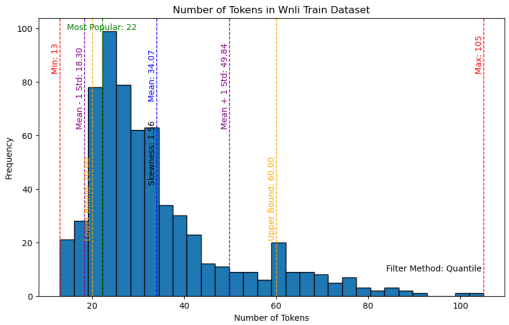
(d) QNLI Dataset



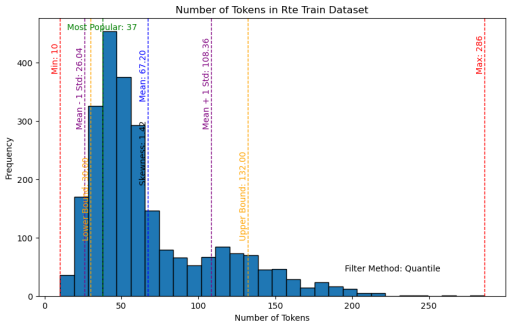
(e) QQP Dataset



(f) SST-2 Dataset

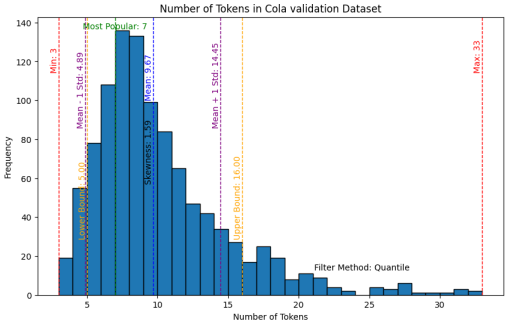


(g) WNLI Dataset

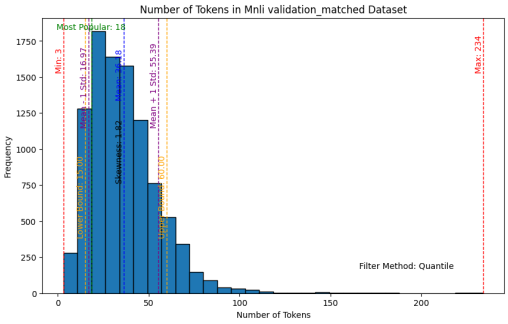


(h) RTE Dataset

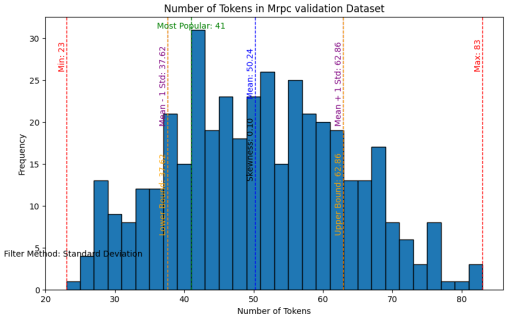
Figure 3. *Data Distribution for train set leveraging the bert-base-uncased tokenizer*



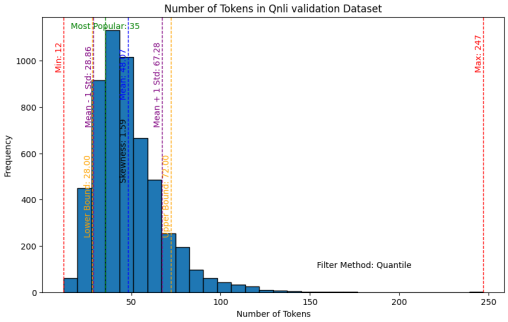
(a) CoLA Dataset



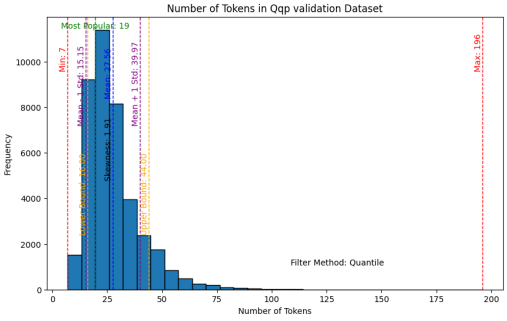
(b) MNLI Dataset



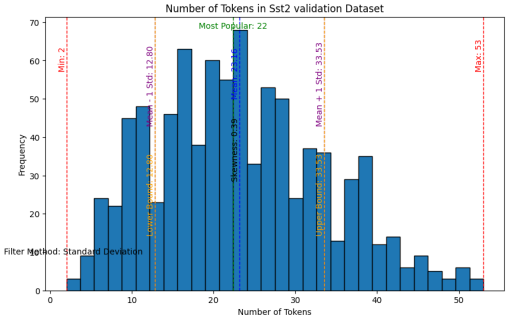
(c) MRPC Dataset



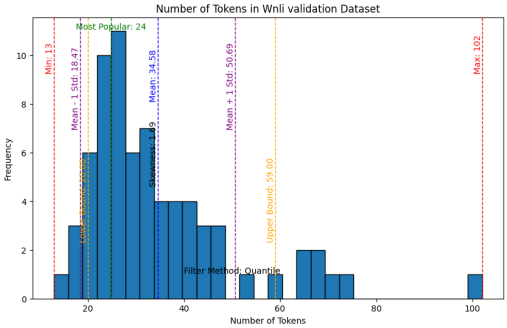
(d) QNLI Dataset



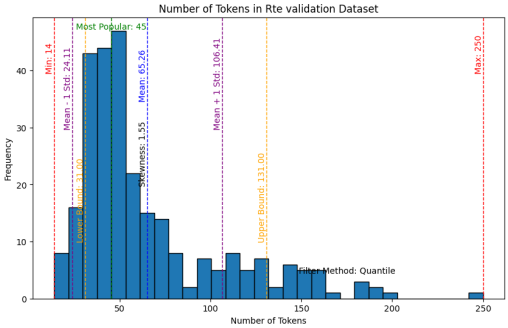
(e) QQP Dataset



(f) SST-2 Dataset

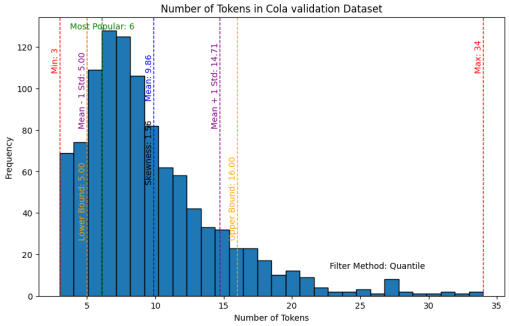


(g) WNLI Dataset

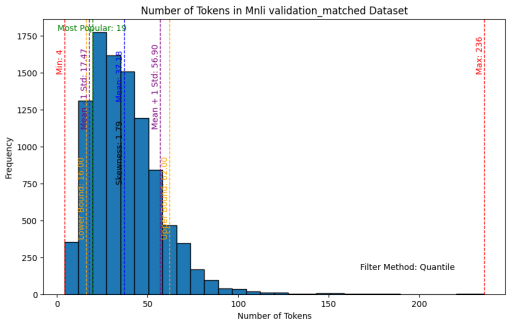


(h) RTE Dataset

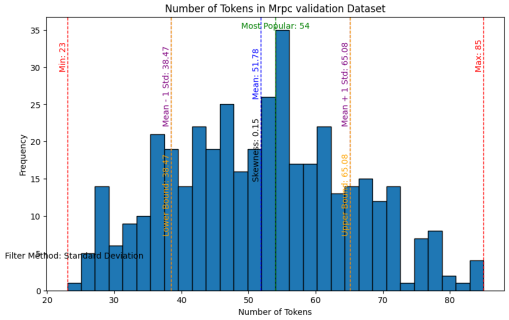
Figure 4. *Data Distribution for validation set leveraging the bert-base-uncased tokenizer*



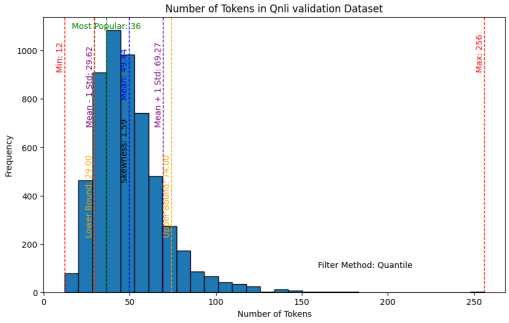
(a) CoLA Dataset



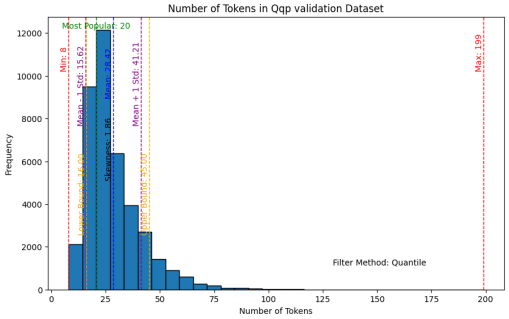
(b) MNLI Dataset



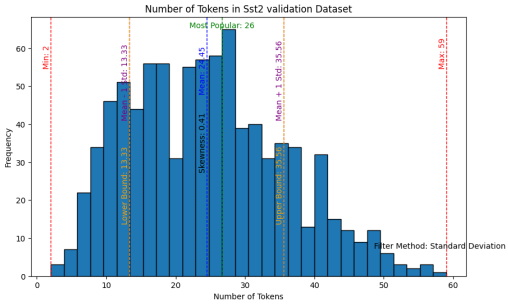
(c) MRPC Dataset



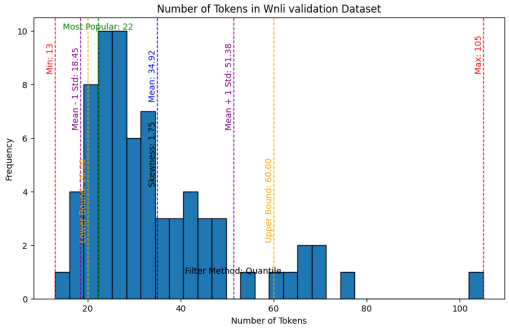
(d) QNLI Dataset



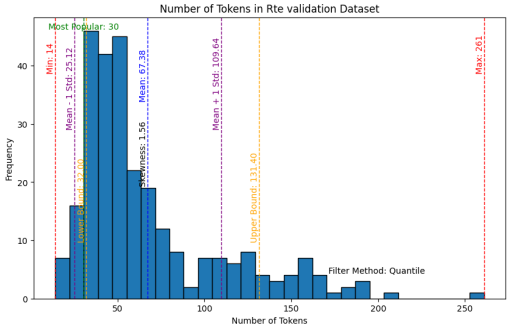
(e) QQP Dataset



(f) SST-2 Dataset



(g) WNLI Dataset



(h) RTE Dataset

Figure 5. *Data Distribution for validation set leveraging the bert-base-cased tokenizer*

.3 Pruning Details

For the baseline algorithms, we have used the $head_mask$ attribution of Transformers library, during metric calculation and the evaluation.

For the algorithm "Multiple Importance LTH", we have used *head mask*, during metric calculation and before fine tuning, we actually pruned the attention heads. When someone actually prunes the heads, the space dimensionality, a layer projects the sample, is reduced. If all the heads of a layer are removed, then the output of the attention mechanism has zero-dimension and thus zero results. However, due to the MLP's bias, the output of the layer is non-zero. (this concerns the **Transformers Library implementation of BERT architecture**)

We tried to fine tune the model, and mask the heads during training, but the results were a lot worse.

List of Abbreviations

ANN	Artificial Neural Network
LLM	Large Language Model
LA	Linguistically Acceptable sentences
LUA	Linguistically Unacceptable sentences
CPT	Constituency Parse Trees
LIG	Layer Integrated Gradients
CIA	Causative-Inchoative Alternation
RAA	Reflexive Antecedent Agreement
SVA	Subject-Verb Agreement
SVO	Subject-Verb-Object
WHE	Wh-Extraction
IG	Integrated Gradients
SIG	Sequential Integrated Gradients
IGSPF	Integrated Gradients Structured Pruning Fine-tuned
IGSPP	Integrated Gradients Structured Pruning Pre-trained
ISP	Iterative Structured Pruning