National Technical University of Athens
MSc in Data Science and Machine Learning
School of Electrical and Computer Engineering
Department of Computer Science
Artificial Intelligence and Learning Systems Laboratory (AILS Lab)
Zografou 157 73, Athens

Adversarial Fine-Tuning of Pretrained Language Models

MSc Thesis

of

GEORGIOS VERNIKOS

Advisor: Andreas-Georgios Stafylopatis
Professor NTUA

Artificial Intelligence and Learning Systems Laboratory (AILS Lab)
Athens, July 2020

National Technical University of Athens
MSc in Data Science and Machine Learning
School of Electrical and Computer Engineering
Department of Computer Science
Artificial Intelligence and Learning Systems Laboratory (AILS Lab)
Zografou 157 73, Athens

Adversarial Fine-Tuning of Pretrained Language Models

MSc Thesis

of

GEORGIOS VERNIKOS

Advisor: Andreas-Georgios Stafylopatis
Professor NTUA

Approved by the commitee on 7 July, 2020.

(Signature)	*(Signature)*	*(Signature)*
.....	.....	.....
Andreas-Georgios Stafylopatis	Giorgos Stamou	Georgios Siolas
Professor	Associate Professor	Teaching and Research Associate
NTUA	NTUA	NTUA

Athens, July 2020

(Signature)

.....
Georgios Vernikos
Electrical & Computer Engineer

Copyright © Georgios Vernikos, 2020.
All rights reserved.

This work is copyright and may not be reproduced, stored nor distributed in whole or in part for commercial purposes. Permission is hereby granted to reproduce, store and distribute this work for non-profit, educational and research purposes, provided that this work and its corresponding publications are acknowledged. Enquiries regarding use for profit should be directed to the author.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the National Technical University of Athens.

To my family.

Acknowledgements

First of all, I would like to thank my supervisor, Prof. Georgios-Andreas Stafylopatis for his useful guidance. I would also like to thank Mr. Georgios Siolas who has assisted me every time I needed him. His help has been crucial even beyond the scope of this thesis.

I would also like to express my gratitude to my family. They have been always motivating me to expect more from myself and never settle. Their expectations have been guiding me to aim for the best. Even though at times I think that we are very different, I know that they have had a huge impact in my personality. I often tend to forget how supportive and patient they have been throughout all these years.

In addition I would like to thank my friends that have in some way helped me during this journey. I have to personally thank my friend Katerina, who has guided and supported me in more ways than I could imagine for the past years. I would also like to thank Ioanna, who had to put up with the full spectrum of my personality and has somehow never lost faith in me. Finally, I would like to thank a group of people with whom I have shared experiences that have changed my perception in multiple ways such as Giorgos, Chara, Nikos, Christina and Antonis. Thank you for being there.

Georgios Vernikos,
July 2020

Abstract

In this work, we investigate methods in order to effectively transfer knowledge from a pretrained model to downstream tasks. Our goal is to improve the performance of pretrained language models on natural language processing tasks. We evaluate our approach on four natural language understanding tasks: sentence acceptability, sentiment analysis, paraphrase detection and textual entailment.

Pretrained language models have achieved state-of-the-art results on most natural language processing tasks. Part of this success lies in their pretraining on large unlabeled corpora. The transferring process of language models consists of further training (fine-tuning) on the task-specific dataset. We argue that this process can hinder the knowledge captured during pretraining, a phenomenon known as *catastrophic forgetting*. In our work, we identify the emergence of too domain-specific features during fine-tuning as a form of catastrophic forgetting. We aim to effectively transfer the knowledge gained during the pretraining of language models to the adaptation process.

In order to preserve most of the knowledge captured during pretraining and exploit the capabilities of pretrained language models, we introduce AFTER, short for domain adversarial fine-tuning as an effective regularizer. We leverage unlabeled data from a different domain than the task-specific dataset and we constrain the extent to which the model representations are allowed to differ across different domains. This constraint is realized through a classifier that tries to distinguish between domains. The parameters of the pretrained language model are trained adversarially to maximize the loss of this classifier. The addition of this domain adversarial loss has a regularizing effect on the fine-tuning process, encouraging domain-invariant representations and subsequently leading to improved performance on downstream tasks.

We experiment with two top-performing pretrained language models (BERT and XLNET), although our approach is equally applicable to any language model. The proposed adversarial fine-tuning method, AFTER, outperforms standard fine-tuning on four natural language understanding tasks using two different pretrained language models. We additionally conduct an ablation study regarding the effect of the domain of origin of the unlabeled corpus and the similarity between the domain of the pretraining and the task-specific data. Our adversarial fine-tuning approach requires minimal changes on the fine-tuning process and leverages unlabeled data.

Keywords

deep learning, natural language processing, natural language understanding, language models, transformers, bert, xlnet, transfer learning, adversarial, fine-tuning, regularizer, sentiment analysis, sentence acceptability, paraphrase detection, textual entailment

Περίληψη

Στην παρούσα διπλωματική εργασία ερευνούμε μεθόδους για τη μεταφορά γνώσης από προεκπαιδευμένα γλωσσικά μοντέλα για την επίλυση προβλημάτων επεξεργασίας φυσικής γλώσσας. Στόχος μας είναι η βελτίωση της απόδοσης προεκπαιδευμένων γλωσσικών μοντέλων σε προβλήματα κατανόησης φυσικής γλώσσας. Προτού παρουσιάσουμε τα επιμέρους υπο-πεδία με τα οποία καταπιανόμαστε σε αυτή την εργασίας κρίνουμε χρήσιμο να εισάγουμε κάποιες βασικές έννοιες του πεδίου της Επεξεργασίας Φυσικής Γλώσσας.

Εισαγωγή

Η γλώσσα αποτελεί ένα δομημένο σύστημα επικοινωνίας το οποίο οι άνθρωποι έχουν εφεύρει και συνεχίζουν να αναπτύσσουν εδώ και αιώνες. Η γλώσσα επιτρέπει στους ανθρώπους να εκφράσουν σκέψεις, πληροφορίες και συναισθήματα. Συνεπώς, η ανάπτυξη υπολογιστικών συστημάτων το οποία μπορούν να χειριστούν την ανθρώπινη γλώσσα είναι μεγάλης σημασίας για την αλληλεπίδραση ανθρώπου-μηχανής. Η επίτευξη αυτού του στόχου έχει οδηγήσει στην παραγωγή μεγάλου όγκου έρευνας στα πεδία της Τεχνητής Νοημοσύνης (Artificial Intelligence), της Υπολογιστικής Γλωσσολογίας (Computational Linguistics) και είναι ο κεντρικός σκοπός της Επεξεργασίας Φυσικής Γλώσσας (Natural Language Processing). Η επεξεργασία φυσικής γλώσσας αποσκοπεί στη γεφύρωση του χάσματος μεταξύ της φυσικής γλώσσας, την οποία χρησιμοποιούν οι άνθρωποι για να επικοινωνήσουν και της αριθμητικής αναπράστασης, η οποία χρησιμοποιείται από τους υπολογιστές για την κωδικοποίηση και επεξεργασία πληροφοριών.

Η ικανότητα οποιασδήποτε οντότητας να καταλάβει και να επεξεργαστεί τη γλώσσα απαιτεί δεξιότητες όπως συλλογιστική και επίλυση προβλημάτων, οι οποίες μπορούν να αποτελέσουν ενδείξεις νοημοσύνης. Με βάση αυτό τον ισχυρισμό η επεξεργασία φυσικής γλώσσας κατηγοριοποιείται ως ένα υπο-πεδίο της τεχνητής νοημοσύνης. Τα πρώτα συστήματα γλωσσικής επεξεργασίας σχεδιάστηκαν με χρήση κανόνων που όρισαν ειδικοί (π.χ. γλωσσολόγοι), με στόχο την μοντελοποίηση βασικών χαρακτηριστικών της γλώσσας όπως η γραμματική και το συντακτικό. Παρ΄ όλα αυτά, τέτοιου είδους συστήματα ήταν απαιτητικά στην κατασκευή και δεν είχαν τη δυνατότητα να γενικέυσουν, να ανταπεξέλθουν δηλαδή σε δείγματα τα οποία δεν είχαν συναντήσει κατά την εκπαίδευσή τους, περιορίζοντας έτσι κατά πολύ τη χρησιμότητά τους.

Το επόμενο κύμα συστημάτων επεξεργασίας φυσικής γλώσσας χρησιμοποιούσαν στατιστικά μοντέλα και βασίστηκαν στη χρήση στατιστικών υποθέσεων και χαρακτηριστικών που δημιουργούσαν οι σχεδιαστές τους, σύμφωνα με το πρότυπο της κλασικής Μηχανικής Μάθησης (Machine Learning). Τα στατιστικά αυτά μοντέλα μπορούσαν να ανιχνεύσουν πολύπλοκα πρότυπα από μεγάλο όγκο δεδομένων χωρίς την απαίτηση για ορισμό συγκεκριμένων κανόνων σχετικών με την επίλυση των επιμέρους προβλημάτων. Αυτό οδήγησε στη δημιουργία γλωσσικών συστημάτων με μεγαλύτερη ακρίβεια και ευρωστία σε τυχόν εισόδους τα οποία μπορούσαν να μοντελοποιήσουν πιο αποδοτικά την ανθρώπινη γλώσσα.

Η πρόσφατη επιτυχία της Βαθιάς Μάθησης (Deep Learning), ενός υπο-πεδίου της Μηχανικής Μάθησης, εδραίωσε τα βαθιά νευρωνικά δίκτυα (Deep Neural Networks) ως τον πιο κοινό αλγόριθμο μάθησης και κατα συνέπεια επηρέασε και τον τομέα της επεξεργασίας φυσικής γλώσσας. Έτσι, τα σύγχρονα συστήματα επεξεργασίας φυσικής γλώσσας αξιοποιούν τις δυνατότητες των βαθιών νευρωνικών δικτύων τα οποία μπορούν να ¨μάθουν' αυτόματα μια ιεραρχία από αναπαραστάσεις-έννοιες χωρίς την ανάγκη για δημιουργία χαρακτηριστικών από τους σχεδιαστές τους. Αυτό είχε ως αποτέλεσμα την ανάπτυξη μοντέλων επεξεργσίας γλώσσας τα οποία επιτυγχάνουν απόδοση συγκρίσιμη με αυτή ενός ανθρώπου σε πολλά προβλήματα κατανόησης γλώσσας. Ο πιο συχνός τρόπος εκπαίδευσης βαθιών νευρωνικών δικτύων, η επιβλεπόμενη μάθηση (supervised learning), απαιτεί την ύπαρξη επισημασμένων δεδομένων τα οποία επιτρέπουν στο εκάστοτε μοντέλο να εξάγει έναν μετασχηματισμό από τα δεδομένα εισόδου στα δεδομένα εξόδου. Όπως είναι φυσικό, η επιτυχία αυτού του τρόπου μάθησης εξαρτάται άμεσα από την ποσότητα των διαθέσιμων επισημασμένων δεδομένων.

Για την αντιμετώπιση αυτού του περιορισμού στα προβλήματα του πραγματικού κόσμου έχουν προταθεί άλλοι τρόποι μάθησης, όπως η μεταφορά μάθησης (transfer learning). Η μεταφορά μάθησης επιλύει το πρόβλημα της περιορισμένης διαθεσιμότητας δεδομένων μεταφέροντας τη γνώση που έχει κατακτηθεί από την επίλυση ενός προβλήματος στη διαδικασία επίλυσης ενός διαφορετικού προβλήματος. Αυτό επιτρέπει στο εκάστοτε μοντέλο να κάνει χρήση της πρότερης γνώσης που κατέκτησε λύνοντας κάποιο αρχικό πρόβλημα, στο οποίο είναι διαθέσιμος ένας μεγάλος όγκος δεδομένων, για την επίλυση ενός νέου προβλήματος. Ο συγκεκριμένος μηχανισμός μάθησης ο οποίος ομοιάζει με τον τρόπο που χτίζεται η γνώση στο ανθρώπινο σύστημα έχει βελτιώσει κατά πολύ την απόδοση και την ικανότητα γενίκευσης των μοντέλων επεξεργασίας γλώσσας και αποτελέι τον κύριο πυλώνα της παρούσας εργασίας.

Γνωστικό Υπόβαθρο

Η παρούσα εργασία αντιμετωπίζει το πρόβλημα της αποδοτικής μεταφοράς μάθησης σε μοντέλα γλωσσικής επεξεργασίας, τα οποία έχουν αρχικά εκπαιδευτεί στο πρόβλημα της μοντελοποίησης γλώσσας (language modelling), για την επίλυση προβλημάτων Κατανόησης Φυσικής Γλώσσας (Natural Language Understanding). Για αυτό τον λόγο θα παρουσιάσουμε συνοπτικά βασικές τεχνικές μηχνανικής μάθησης και επεξεργασίας φυσικής γλώσσας οι οποίες αποτελούν αντικείμενο μελέτης αυτής της εργασίας.

Μεταφορά Μάθησης: Η μεταφορά μάθησης έχει αποτελέσει καταλυτικό παράγοντα για τα πρόσφατα αξιοσημείωτα επιτεύγματα των μοντέλων βαθιάς μάθησης στο πεδίο της επεξεργασίας φυσικής γλώσσας. Το σχήμα μεταφοράς μάθησης το οποίο μελετάται σε αυτή την εργασία είναι η ακολουθιακή μεταφορά μάθησης (sequential transfer learning), όπου το μοντέλο αρχικά εκπαιδεύεται (προεκπαιδεύεται) για την επίλυση ενός προβλήματος και στη συνέχεια εκπαιδεύεται περαιτέρω (προσαρμόζεται) για να λύσει ένα διαφορετικό πρόβλημα. Η αποτελεσματικότητα αυτής της μεθόδου έγκειται στο γεγονός ότι, με κατάλληλη επιλογή του αρχικού προβλήματος, κατά την προεκπαίδευση χρησιμοποείται μια πληθώρα δεδομένων και δημιουργούνται γενικές αναπραστάσεις από το μοντέλο. Συνεπώς, το προεκαπιδευμένο μοντέλο αποτελεί μια καλύτερη αρχικοποίηση για το τελικό μοντέλο. Μέσω κατάλληλης διαδικασίας προσαρμογής, η ακολουθιακή μεταφορά μάθησης οδηγεί σε ευρύτερη γενίκευση, αυξημένη ταχύτητα σύγκλισης και καλύτερα αποτελέσματα στο εκάστοτε τελικό πρόβλημα. Η διαδικασία προσαρμογής, όμως, μπορεί να οδηγήσει σε απώλεια των πρότερων, γενικών αναπαραστάσεων που δημιούργησε το μοντέλο κατά την αρχική του εκπαίδευση. Αυτό το φαινόμενο είναι γνωστό ως καταστροφική λησμόνηση (catastrophic forgetting) και αποτελεί κύρια πρόκληση στη διαδικασία μεταφοράς μάθησης.

Μια επιπλέον εκδοχή της μεταφοράς μάθησης που χρησιμοποιείται στην παρούσα εργασία

είναι η ταυτόχρονη μάθηση πολλών εργασιών (multitask learning) όπου το μοντέλο εκπαιδεύεται ταυτόχρονα για την επίλυση δύο ή περισσότερων προβλημάτων. Αυτή η διαδικασία μάθησης συνήθως οδηγεί στη δημιουργία αναπαραστάσεων από το μοντέλο οι οποίες είναι χρήσιμες για παραπάνω από ενα πρόβλημα καθώς το μοντέλο επωφελέιται από την παράλληλη επίλυση των επιμέρους προβλημάτων. Η ταυτόχρονη μάθηση, πέρα από το προφανές πλεονέκτημα της μείωσης του χρόνου για την εκμάθηση των προβλημάτων έχει ως αποτέλεσμα τη βελτίωση της ευρωστίας και της ικανότητας γενίκευσης του μοντέλου το οποίο κατα συνέπεια οδηγεί σε αυξημένη απόδοση.

Επίσης, η μέθοδος προσαρμογής που προτείνεται σε αυτή την εργασία είναι εμπνευσμένη από την προσαρμογή σε συγκεκριμένο τομέα (domain adaptation). Σε αυτή την περίπτωση μεταφοράς μάθησης το αρχικό πρόβλημα είναι το ίδιο με το τελικό πρόβλημα, που αποτελεί το στόχο της μεθόδου, αλλά η κατανομή των δεδομένων εκπαίδευσης για τα δύο προβλήματα είναι διαφορετική. Η ικανότητα γενίκευσης του μοντέλου σε άγνωστα (που δεν συμπεριλαμβάνονται στα δεδομένα εκπαίδευσης) δεδομένα είναι καταλυτική για την επιτυχημένη προσαρμογή σε διαφορετικό τομέα. Εμπνεόμαστε από αυτή την υπόθεση για να προτείνουμε μια επέκταση στη διαδικασία προσαρμογής προεκπαιδευμένων μοντέλων.

Μοντελοποίηση Γλώσσας: Η μοντελοποίησης γλώσσας είναι το πρόβλημα που χρησιμοποιείται πιο συχνά για την προεκπαίδευση των μοντέλων γλωσσικής επεξεργασίας στη διαδικασία ακολουθιακής μάθησης. Ο στόχος της μοντελοποίησης γλώσσας είναι η εκτίμηση της κατανομής των λέξεων για οποιαδήποτε ακολουθία λεξεων που συναντάται στη γλώσσα. Αυτό το πρόβλημα μοντελοποιείται ως εξής: πρόβλεψη της ν-οστής λέξης, δεδομένων των ν-1 λέξεων μιας ακολουθίας. Από αυτό τον ορισμό είναι εμφανές ότι ένας από τους λόγους για την προτίμηση της μοντελοποίησης γλώσσας έναντι άλλων προβλημάτων για την προεκπαίδευση των μοντέλων επεξεργασίας γλώσσας είναι το γεγονός ότι δεν απαιτεί επισημασμένα δεδομένα. Ένας ακόμα λόγος είναι ότι η γλωσσική μοντελοποίηση σαν πρόβλημα συνεπάγεται με την μάθηση διάφορων γνωρισμάτων της γλώσσας. Τα μοντέλα τα οποία εκπαιδεύονται σε αυτό το πρόβλημα, δηλαδή τα Γλωσσικά Μοντέλα (Language Models), αποτελούν μια καλή αρχικοποίηση για την κατασκευή μοντέλων με στόχο την επίλυση πολλών προβλημάτων επεξεργασίας φυσικής γλώσσας, επιτυγχάνοντας κορυφαία απόδοση.

Πέρα από τον κλασικό φορμαλισμό της μοντελοποίησης γλώσσας ως αυτο-παλινδρόμηση (διότι κάθε έξοδος εξαρτάται από τις προηγούμενες εξόδους και έναν στοχαστικό όρο), εναλλακτικοί φορμαλισμοί έχουν προταθεί. Δύο από τους διάφορους φορμαλισμούς που έχουν προταθεί είναι η Μοντελοποίησης Γλώσσας με χρήση Μάσκας (Masked Language Modelling) και η Μοντελοποίηση Γλώσσας με χρήση Μεταθέσεων (Permutation Language Modelling). Ο πρώτος φορμαλισμός ορίζει το πρόβλημα ως την πρόβλεψη κ λέξεων σε μια ακολουθία δεδομένων όλων των ν-κ που προηγούνται και έπονται αυτών. Αυτός ο φορμαλισμός έχει το πλεονέκτημα ότι χρησιμοποιεί το περιεχόμενο της ακολουθίας και από τις δύο κατευθύνσεις (από αριστερά προς τα δεξιά και από τα δεξία προς τα αριστερά), οδηγώντας σε βελτιωμένη απόδοση. Όμως με αυτό το φορμαλισμό οι μάσκες εμφανίζονται μόνο κατά την επίλυση αυτού του προβλήματος και επίσης οι προβλέψεις για τις κ λέξεις γίνονται ανεξάρτητα, δηλαδή κάθε πρόβλεψη είναι ανεξάρτητη από τις άλλες. Για την αντιμετώπιση αυτών των προβλημάτων έχει προταθεί η μοντελοποίηση γλώσσας με χρήση μεταθέσεων η οποία προβλέπει την ν-οστή λέξη με βάση της ν-1 λέξεις που προηγούνται αυτής σε κάθε μία πιθανή μετάθεση των λέξεων. Αυτός ο φορμαλισμός συνδυάζει το περιεχόμενο και από τις δύο κατευθύνσεις χωρίς τις υποθέσεις του πρώτου φορμαλισμού.

Προβλήματα Κατανόησης Γλώσσας: Τα προβλήματα κατανόησης γλώσσας προσπαθούν να αντιμετωπίσουν την πρόκληση της ερμηνείας δεδομένων σε μορφή γλώσσας. Το πιο γνωστό πρόβλημα κατανόησης γλώσσας είναι η ανάλυση συναισθήματος (sentiment analysis) το οποίο εκφράζεται σε ένα κείμενο. Η ανάλυση συναισθήματος είναι ένα πρόβλημα κατη-

γοριοποίησης σε δύο (θετικό/αρνητικό) ή περισσότερες κλάσεις. Η ανάλυση συναισθήματος έχει πολλές πρακτικές εφαρμογές στον τομέα της διαφήμισης, της πολιτικής και της αλληλεπίδρασης ανθρώπου-μηχανής. Ένα ακόμα πρόβλημα κατανόησης γλώσσας που αντιμετωπίζουμε σε αυτή την εργασία είναι το πρόβλημα της καταλληλότητας μιας πρότασης (sentence acceptability) όπου στόχος είναι η κατηγοριοποίηση της εκάστοτε πρότασης ως γραμματικά αποδεκτή ή μη αποδεκτή. Άλλο ένα πρόβλημα κατανόησης γλώσσας είναι η ανίχνευση παράφρασης (paraphrase detection). Σε αυτό το πρόβλημα, παρατίθενται δύο προτάσεις και εξετάζεται η πιθανότητα η μία να είναι παράφραση της άλλης πρότασης. Τέλος, σε αυτή τη διπλωματική εργασία ασχολούμαστε και με το πρόβλημα της ανίχνευσης επαγωγής σε κείμενο (textual entailment), το οποίο αφορά και πάλι ένας ζεύγος προτάσεων. Στόχος αυτού το προβλήματος είναι ο προσδιορισμός του ενεδεχομένου η πρώτη πρόταση να επάγει ή όχι τη δεύτερη. Όλα τα προβλήματα που αναφέραμε είναι προβλήματα κατηγοριοποίησης όπου ο αριθμός των κλάσεων εξαρτάται από τον ορισμό του εκάστοτε προβλήματος.

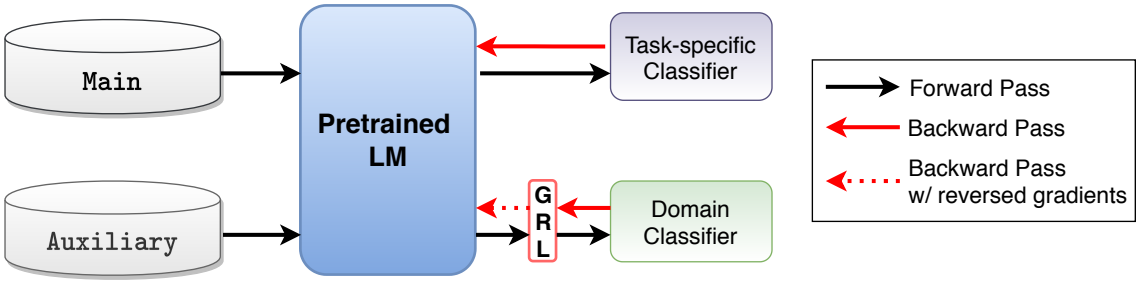
Προσαρμογή Προεκπαιδευμένων Γλωσσικών Μοντέλων σε Προβλήματα Επεξεργασίας Φυσικής Γλώσσας

Στην επεξεργασία φυσικής γλώσσας, προεκπαιδευμένα γλωσσικά μοντέλα τα οποία προσαρμόζονται στο εκάστοτε πρόβλημα έχουν επιτύχει κορυφαία απόδοση τα τελευταία χρόνια. Σε αυτή την εργασία, επεκτείνουμε τον καθιερωμένο τρόπο προσαρμογής προεκπαιδευμένων γλωσσικών μοντέλων εισάγοντας έναν νέο όρο εξομάλυνσης, AFTER; domain Adversarial Fine-Tuning as an Effective Regularizer. Συγκεκριμένα, συμπληρώνουμε τον όρο ποινής που αντιστοιχεί στο εκάστοτε πρόβλημα κατά την προσαρμογή με έναν αντιπαλικό όρο ποινής (adversarial objective). Αυτός ο επιπρόσθετος όρος ποινής σχετίζεται με έναν αντιπαλικό ταξινομητή (adversarial classifier) ο οποίος έχει ως στόχο το διαχωρισμό μεταξύ αναπαραστάσεων που αντιστοιχούν εντός-τομέα (in-domain) και εκτός-τομέα (out-of-domain). Με τον όρο εντός-τομέα αναφερόμαστε στα επισημασμένα δεδομένα του εκάστοτε προβλήματος ενώ με τον όρο εκτός-τομέα εννοούμε μη επισημασμένα δεδομένα που προέρχονται από έναν διαφορετικό τομέα. Διαισθητικά, ο αντιπαλικός ταξινομητής επωμίζεται το ρόλο του ομαλοποιητή ο οποίος αποτρέπει το μοντέλο από το να υπερ-εκπαιδευτεί στον τομέα που αφορά το συγκεκριμένο πρόβλημα. Εμπειρικά αποτελέσματα σε τέσσερα προβλήματα κατανόησης γλώσσας και με δύο διαφορετικά γλωσσικά μοντέλα καταδεικνύουν ότι το AFTER βελτίωνει συστηματικά την απόδοση σε σχέση με τον καθιερωμένο τρόπο προσαρμογής.

Προτεινόμενη Μέθοδος

Στο Σχήμα 1 παρουσιάζεται μια εποπτική απεικόνιση της προτεινόμενης μεθόδου, AFTER. Το κυριώς πρόβλημα (`Main`) που καλούμαστε να λύσουμε περιλαμβάνει ένα επισημασμένο σετ δεδομένων εκπαίδευσης που προέρχονται από ένα συγκεκριμένο τομέα. Επιπλέον χρησιμοποιούμε έναν όγκο μη επισημασμένων βοηθητικών δεδομένων (`Auxiliary`) που προέρχονται από έναν διαφορετικό τομέα. Επισημειώνουμε κάθε δείγμα από τα κυρίως και τα βοηθητικά δεδομένα με βάση τον τομέα προέλευσής τους. Χρησιμοποιούμε αυτές τις ετικέτες για την εκπαίδευση ενός ταξινομητή ο οποίος προσπαθεί να κατηγοριοποιήσει κάθε δείγμα ως προς τον τομέα που ανήκει (εντός/εκτός).

Αρχικοποιούμε το μοντέλο μας με τα βάρη από κάποιο κορυφαίο προεκπαιδευμένο γλωσσικό μοντέλο όπως το BERT ή το XLNET. Για το κυρίως πρόβλημα προσθέτουμε στην ήδη υπάρχουσα αρχιτεκτονική του μοντέλου τον ανάλογο ταξινομητή που δημιουργεί τον όρο ποινής L_{Main}. Επιπλέον, προσθέτουμε ακόμα έναν ταξινομητή ο οποίος προσπαθεί να διαχωρίσει τους τομείς προέλευσης και παράγει τον όρο ποινής L_{Domain}.



Σχήμα 1: Απεικόνιση της προτεινόμενης μεθόδου, AFTER. Ο ταξινομητής που αντιστοιχεί στο εκάστοτε πρόβλημα (task-specific classifier) χρησιμοποιεί τα δεδομένα του προβλήματος (`Main`) ενώ ο αντιπαλικός ταξινομητής τομέα (domain classifier) κάνει χρήση των κυρίως (`Main`) και των βοηθητικών δεδομένων (`Auxiliary`).

Με αυτή τη μέθοδο αναζητούμε αναπαραστάσεις οι οποίες είναι χαρακτηριστικές για το κυρίως πρόβλημα αλλά δεν είναι διαχωρίσιμες όσον αφορά τον τομέα προέλευσης. Για αυτό το λόγο ελαχιστοποιούμε τον όρο ποινής που αφορά το κυρίως πρόβλημα ενώ μεγιστοποιούμε τον όρο ποινής που αφορά την ταξινόμηση ως προς τον τομέα. Οι παράμετροι του ταξινομητή τομέα εκπαιδεύονται έτσι ώστε να προβλέψει τη σωστή ετικέτα ενώ το υπόλοιπο δίκτυο εκπαιδεύεται έτσι ώστε να τον ¨εξαπατήσει¨, δημιουργώντας αναπαραστάσεις που είναι ανεξάρτητες από τον τομέα προέλευσης των δεδομένων. Για αυτή τη διαδικασία χρησιμοποιούμε ένα επίπεδο αντιστροφής κλίσης (`GRL`), όπως φαίνεται στο Σχήμα 1. Το επίπεδο αυτό κατά την εμπροσθοδιάδοση (forward pass) δεν έχει κάποια επιρροή ενώ κατά την οπισθοδιάδοση (backward pass) αντιστρέφει το πρόσημο των εισερχομένων μερικών παραγώγων.

Πειράματα και Αποτελέσματα

Για τα πειράματά μας χρησιμοποιούμε τέσσερα σετ δεδομένων από διάφορα προβλήματα κατανόησης γλώσσας, ως κυρίως προβλήματα. Τα προβλήματα αυτά είναι η ανίχνευση καταλληλότητας πρότασης (CoLA), η ανάλυση συναισθήματος (SST-2), η ανίχνευση παράφρασης (MRPC) και η ανίχνευση επαγωγής σε κείμενο (RTE). Για τα βοηθητικά δεδομένα επιλέγουμε σετ δεδομένων από διάφορετικούς τομείς όπως: νέα (News), κριτικές (Reviews), νομικά κείμενα (Legal), ιατρικά δοκίμια (Medical) και μαθηματικές ερωτήσεις (Math).

Ως πρώτη και δεύτερη μέθοδο αναφοράς χρησιμοποιούμε τον καθιερωμένο τρόπο προσαρμογής του μοντέλου BERT και XLNet, αντίστοιχα. Στον Πίνακα 1 παρατίθενται τα αποτελέσματα για όλους τους συνδυασμούς πειραμάτων (εκτός από το πρόβλημα καταλληλότητας πρότασης για το δεύτερο μοντέλο, λόγω περιορισμών της χρησιμοποιούμενης υποδομής). Παρατηρούμε ότι η προτεινόμενη μέθοδος AFTER βελτιώνει τα αποτελέσματα σε σχέση με τον καθιερωμένο τρόπο προσαρμογής σε όλα τα προβλήματα και για τα δύο προεκπαιδευμένα γλωσσικά μοντέλα. Η βελτίωση στην απόδοση διαφέρει ανάλογα με το εκάστοτε πρόβλημα και μοντέλο. Παρατηρούμε ακόμα ότι η απόδοση αυξάνεται με τη χρήση των περισσότερων βοηθητικών δεδομένων το οποίο αποτελεί ένδειξη ευρωστίας της μεθόδου σε σχέση με την επιλογή αυτών των δεδομένων. Επίσης παρατηρούμε ότι σε ορισμένες περιπτώσεις η προτινόμενη μέθοδος προσαρμογής, AFTER (σε συνδυασμό με ένα αποδοτικό γλωσσικό μοντέλο π.χ. BERT) οδηγεί σε αυξημένη απόδοση σε σχέση με τη χρήση ενός ακόμα πιο βελτιωμένου μοντέλου (π.χ. XLNet). Αυτό μας οδηγεί στο συμπέρασμα ότι η μέθοδος μας μπορεί να εφαρμοστεί σε πολλές αρχιτεκτονικές και μοντέλα για την επίτευξη αυξημένης απόδοσης στο εκάστοτε γλωσσικό πρόβλημα. Συνεπώς, αυτή η σειρά πειραμάτων καταδεικνύει την ανάγκη για αναζήτηση πιο αποδοτικών τρόπων προσαρμογής των υπαρχόντων γλωσσικών μοντέλων, σαν κατεύθυνση έρευνας, παράλληλα με τη συνεχή βελτίωση των γλωσσικών μοντέλων.

	CoLA	**SST-2**	**MRPC**	**RTE**
	Matthews corr.	*Accuracy*	*Accuracy / F1*	*Accuracy*
BERT	55.5 ± 3.2	92.0 ± 0.5	85.4 ± 1.1 / 89.6 ± 0.6	64.3 ± 3.1
AFTER W/ NEWS	$\mathbf{57.3} \pm 1.5$	92.5 ± 0.4	$\mathbf{87.5} \pm 1.7$ / $\mathbf{91.1} \pm 1.2$	64.7 ± 1.9
AFTER W/ REVIEWS	57.1 ± 1.2	92.4 ± 0.3	86.4 ± 0.3 / 90.1 ± 0.4	64.6 ± 0.8
AFTER W/ LEGAL	55.0 ± 1.5	92.4 ± 0.3	86.6 ± 0.6 / 90.3 ± 0.5	$\mathbf{64.8} \pm 1.9$
AFTER W/ MEDICAL	55.9 ± 2.9	$\mathbf{92.6} \pm 0.3$	86.9 ± 1.3 / 90.7 ± 1.0	62.6 ± 3.4
AFTER W/ MATH	56.1 ± 2.8	92.3 ± 0.8	87.3 ± 0.9 / 90.8 ± 0.7	62.5 ± 1.3
XLNET	–	93.0 ± 0.7	86.4 ± 0.7 / 90.1 ± 0.5	64.7 ± 4.4
AFTER W/ NEWS	–	$\mathbf{93.9} \pm 0.3$	87.3 ± 0.7 / 91.0 ± 0.5	63.9 ± 2.3
AFTER W/ REVIEWS	–	93.5 ± 0.3	86.9 ± 0.6 / 90.5 ± 0.5	65.1 ± 2.8
AFTER W/ LEGAL	–	93.6 ± 0.5	$\mathbf{87.5} \pm 1.6$ / $\mathbf{90.9} \pm 1.2$	64.8 ± 1.6
AFTER W/ MEDICAL	–	93.3 ± 0.5	87.0 ± 1.1 / 90.5 ± 0.7	64.5 ± 2.1
AFTER W/ MATH	–	$\mathbf{93.9} \pm 0.4$	87.3 ± 1.2 / 90.8 ± 0.9	$\mathbf{66.1} \pm 1.9$

Πίνακας 1: Σύγκριση του καθιερωμένου τρόπου προσαρμογής και του AFTER για το BERT (Πάνω) και το XLNET (Κάτω). Τα υπογραμμισμένα αποτελέσματα ξεπερνούν τη μέθοδο αναφοράς. Τα καλύτερα αποτελέσματα απεικονίζονται με **έντονη** γραφή. Παρουσιάζουμε τη μέση τιμή και τη διακύμανση σε πέντε εκτελέσεις του κάθε πειράματος.

Συμπεράσματα

Σε αυτή την εργασία προτείνουμε το AFTER, μια εναλλακτική μέθοδο προσαρμογής προεκπαιδευμένων γλωσσικών μοντέλων με βάση ένα αντιπαλικό όρο ποινής που σχετίζεται με την προσαρμογή τομέα. Διαισθητικά, αυτός ο όρος περιορίζει την απώλεια της γνώσης που αποκτήθηκε από το μοντέλο κατά την προεκπαίδευση οδηγώντας σε πιο αποδοτική προσαρμογή. Τα αποτελέσματα από τον πειραματισμό μας με δυο διαφορετικά γλωσσικά μοντέλα αναδεικνύουν ότι η μέθοδος μας βελτιώνει συστηματικά την απόδοση σε σχέση με τον καθιερωμένο τρόπο προσαρμογής. Η μέθοδος μας είναι εφαρμόσιμη σε διάφορα μοντέλα και αρχιτεκτονικές, με ελάχιστες τροποποιήσεις στη διαδικασία προσαρμογής και με τη χρήση επιπλέον μη επισημασμένων δεδομένων. Το δεύτερο χαρακτηριστικό είναι ιδιαίτερα σημαντικό καθώς τα μη επισημασμένα δεδομένα είναι ευρέως διαθέσιμα από διάφορες πηγές (όπως και αυτά που χρησιμοποιήθηκαν στην παρούσα εργσία). Η προτινόμενη μέθοδος μπορεί να εφαρμοστεί στην προσαρμογή οποιουδήποτε μοντέλου, πέρα από τα γλωσσικά μοντέλα. Τέλος, σε μελλοντική επέκταση της παρούσας ερευνητικής εργασίας θα θέλαμε να εξερευνήσουμε την εφαρμογή της μεθόδου για την προσαρμογή σε διαφορετικές γλώσσες, γενικεύοντας την έννοια του τομέα, για την απόκτηση αναπαραστάσεων που είναι ανεξάρτητες της εκάστοτε γλώσσας.

Λέξεις κλειδιά
βαθιά μάθηση, επεξεργασία φυσικής γλώσσας, κατανόηση φυσικής γλώσσας, μεταφορά μάθησης, αντιπαλικός όρος ποινής, προσαρμογή, εξομαλυντής, ανάλυση συναισθήματος, ανίχνευση επαγωγής σε κείμενο, ανίχνευση παράφρασης, γλωσσικά μοντέλα

Contents

List of Figures

List of Tables

Notation

Probability and Statistics

P	Probability mass function
p	Probability density function
σ	Standard deviation
σ^2	Variance
$\mathcal{N}$	Gaussian distribution

Machine Learning

$\mathcal{T}$	Task
$\mathcal{D}$	Domain
$\mathbf{W}$	Weight matrix
X	Set of training examples
Y	Set of training labels
$\mathcal{L}$	Loss function
η	Learning rate
$\boldsymbol{\theta}$	Model parameters

We use the following conventions throughout this thesis: We denote real, integer and natural numbers as $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{N}$, respectively. Scalars are represented by no-boldface letters (x), vectors appear in boldface lowercase letters ($\mathbf{x}$) and matrices are indicated by boldface uppercase letters ($\mathbf{A}$). All vectors are assumed to be column vectors unless they are explicitly defined as row vectors. We use subscripts with bold letters ($\mathbf{x}_i$) to refer to entire rows or columns and subscripts and letters for specific elements (x_i). The concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted with $\mathbf{a}||\mathbf{b}$ and in some cases with $[\mathbf{a};\mathbf{b}]$. The Hadamard product (element-wise multiplication) of vectors $\mathbf{a}$ and $\mathbf{b}$ is denoted with $\mathbf{a} \odot \mathbf{b}$. For the matrices $\mathbf{A} \in \mathbb{R}^{n \times m}$, $\mathbf{B} \in \mathbb{R}^{n \times m}$ we indicate their Hadamard product as $\mathbf{A} \otimes \mathbf{B}$.

Chapter 1

Introduction

1.1 Motivation

Language is a structured system for communication that has been employed and developed by humans for thousands of years. Language enables humans to communicate information and express thoughts and sentiments. Consequently, the development of computer systems that can handle human language is of great importance for human-computer interaction. This objective has fueled a considerable volume of research in Artificial Intelligence (AI) and Computational Linguistics and is the primary goal of Natural Language Processing (NLP). Natural Language Processing aims to bridge the gap between human communication, via *natural language* and numeric representation, that is used by computer systems in order to encode and process information.

The ability of an entity to understand and generate language requires reasoning and problem solving skills, which, consequently, demonstrate a form of intelligence. This requirement legitimizes natural language processing as a subfield of AI. The first language-processing computer systems were designed based on rules created by human experts, that formally described aspects of language such as grammar and syntax. However, these systems were difficult to construct and did not *generalize*; they were not capable of efficiently processing previously unseen inputs. These drawbacks severely limited the applicability of the first language processing systems.

The next wave of NLP systems was based on statistical models that leveraged statistical hypotheses and handcrafted features in order to automatically infer patterns, useful for language processing, according to the Machine Learning (ML) paradigm. In this case, the human effort was concentrated in the feature engineering process instead of the creation of rules. The partial automation of language learning and the use of statistical inference algorithms that can leverage large amounts of data, produced models that were more accurate and robust to unfamiliar inputs.

The recent success of Deep Learning (DL) [58], [107], a subfield of ML, established Deep Neural Networks (DNNs) as the most common learning algorithm and consequently influenced the field of NLP. As a result, most state-of-the-art language processing systems leverage the power of deep neural networks that are able to learn a hierarchy of concepts, significantly limiting the requirement for feature engineering. This has lead to NLP models that achieve near-human performance [28], [69], [96] on a set difficult language understanding tasks [119]. The learning process of DNNs, for most NLP tasks, involves the use of annotated datasets that enable the model to infer a mapping from the input data to the output data. However, the success of this learning paradigm, called supervised learning, is constrained by the amount of available annotated data.

In order to handle the limited availability of annotated data in real-world scenarios, alternative learning schemes are employed. One alternative is *transfer learning*, which is explored throughout this thesis. Transfer learning addresses the challenge of limited annotated data by transferring the knowledge acquired from solving one task to the process of solving a different task. This allows the model to leverage prior knowledge that was gained through solving a task that is provided with a large amount of data in order to solve a specific task with limited annotated data. This learning scheme, which is similar to the way humans build up knowledge, has significantly improved performance and generalization ability of NLP models [28], [51], [94] and is the main focus of this work.

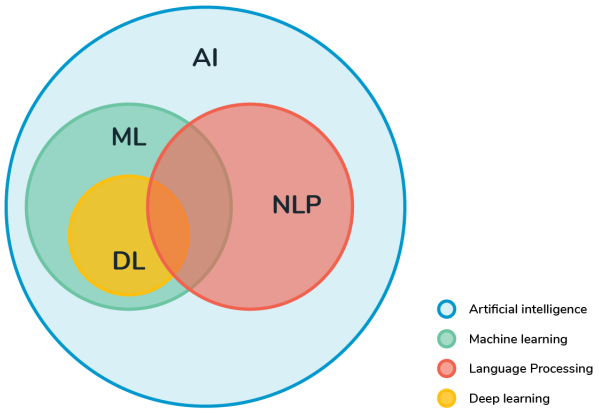


Figure 1.1: Relationship between NLP, AI and ML/DL. Figure from Learn Natural Language Processing from scratch - Good Audience blog.

To this end, we aim to contribute to research methods that leverage prior knowledge, gained by solving other tasks, in order to solve a specific task more efficiently. In our research we experiment with state-of-the-art deep learning models for natural language processing tasks and propose methods in order to improve their performance on these tasks. Although we experiment with a number of NLP tasks the proposed transfer learning framework is applicable to various other NLP tasks. Hence, this work lies in the intersection of NLP with ML and DL, as illustrated in Figure 1.1.

1.2 Research Objectives & Contributions

This thesis addresses the problem of efficiently transferring knowledge from NLP models that were petrained on the language modelling task in order to solve natural language understanding tasks. We will therefore introduce the following areas of ML and NLP that are studied in this work:

Transfer Learning: Transfer learning [84] has been a crucial factor of the recent remarkable achievements of deep learning models in the field of NLP [26], [28], [51], [96]. A transfer learning scenario that is studied in this work is *sequential transfer learning* [97], where a model is initially trained (*pretrained*) to solve a source task and then is further trained (*fine-tuned*) to solve a different target task. The efficiency of this method lies in the fact that the pretrained model leverages a plethora of additional data and creates generic representations due to the particular choice of the pretraining task. The pretrained model, thus, serves as a better initialization for the final model. With proper fine-tuning, sequential transfer learning results in better generalization, faster convergence

and improved performance. However, the fine-tuning process often results in loss of the initial, generic representations that were captured by the model during pretraining. This situation is known as *catastrophic forgetting* [43] and is a central challenge in transfer learning.

Another transfer learning scenario that is also employed in this work is *multitask learning*, where two or more tasks are learned simultaneously. This transfer learning scheme results in the emergence of representations that are useful for all the tasks, since the model benefits from solving multiple tasks at the same time. This enhances the generalization and robustness of the model that can subsequently lead to improved performance.

Furthermore, the fine-tuning approach that is proposed in this work is inspired by *domain adaptation*. In this transfer learning setting the initial source task is the same as the final target task but the distribution of data samples are different. The performance of a model on unseen data is crucial for the success of domain adaptation. We are inspired by this requirement in order to propose an extension to the standard fine-tuning technique.

Language Modelling: Language modelling is the most common pretraining task for sequential transfer learning in NLP. The goal of language modelling is to estimate the probability distribution of words for any given word sequence. This task is formulated as the prediction of the nth word given the previous $n-1$ words in the sequence. From this formulation, it is evident that one of the reasons for the popularity of language modelling as a pretraining task is the fact that it can leverage unannotated data. Another reason, is the fact that language modelling captures many aspects of the structure of language [93], [132]. The models that are trained to solve this task, the language models (LMs), have been reported to be a good initialization point for various NLP tasks [51], [94], achieving state-of-the-art performance.

Beyond the classic definition of language modelling as autoregression (since each output depends on the previous output and a stochastic term), alternative formulations have been proposed. Two of the various formulations that have been suggested is *Masked Language Modelling* and *Permutation Language Modelling*. In the first modelling formulation, the problem is defined as the prediction of k masked words, given all the $n-k$ words that come before or after these words in the sequence. This definition of language modelling enables the models to leverage bidirectional context (from left-to-right and right-to-left), leading to improved performance. However, the masks only occur during the training for this task and the predictions of the k masked words are independent (each masked word is predicted independent of the others). In order to address these limitations, permutation language modelling aims to predict the nth word given the previous $n-1$ words that precede this word in any possible permutation of the word sequence. This formulation combines bidirectional context without the independence assumptions of masked language modelling.

Natural language Understanding tasks: Natural language Understanding (NLU) tasks aim to tackle the challenge of interpreting natural language input. A common NLU tasks is sentiment analysis that aims to identify the sentiment that is expressed in a text. A basic form of sentiment analysis is the classification of the polarity of a text as being positive or negative (or neutral). This case could be extended with multiple classes that express different sentiments. Sentiment analysis is an NLP task with many practical applications such marketing, advertising, politics and human-machine interaction systems, such as chatbots. Other NLU tasks that are addressed in this work are sentence acceptability, paraphrase detection and textual entailment.

In this work we propose a different technique for fine-tuning pretrained language models on downstream tasks. We argue that the standard fine-tuning technique hurts the generic

representations that are captured by the model during pretraining, which consequently has a negative effect in performance. We therefore propose a fine-tuning technique that allows the model to adapt to a specific task and domain, by modifying its initial representations. Nevertheless, at the same time, we aim to regularize the fine-tuning process, by adding an auxiliary loss. We identify the emergence of too domain-specific representation during fine-tuning as a form of catastrophic forgetting and constrain the extent to which the fine-tuned representations of the model are allowed to differ across domains. The proposed approach does not require additional labeled data and can be integrated in the fine-tuning process with minimal changes to the model. Furthermore, the suggested method outperforms the standard fine-tuning of a state-of-the-art pretrained LM in various NLU tasks, while it can be easily applied to other tasks as well.

1.3 Thesis Outline

Chapter 2 provides the background knowledge in ML that is essential in order to follow this thesis. We review the fundamental learning paradigms of Machine Learning that are employed in this work. We do not introduce classic ML algorithms that are not employed in this work. Instead, we delve deeper into the ML models that are used in this work, deep neural networks. Therefore, we analyse the deep learning models and techniques that are employed in this work.

Chapter 3 presents the field of natural language processing, introducing the background that is needed to set the environment in which NLP is developed. After the introduction of the tasks that are addressed in this thesis, the current deep learning techniques for NLP are presented. We additionally analyse the NLP models that are used in this thesis.

Chapter 4 includes a detailed analysis of our work, *domain Adversarial Fine-Tuning as an Effective Regularizer*. We first introduce the challenges and limitations of the standard fine-tuning process. We then present adversarial training for domain adaptation and propose our domain adversarial fine-tuning approach. We demonstrate our empirical results in different NLP tasks and pretrained models and, also, make an ablation study to explore the efficiency of our method.

Chapter 5, finally, contains our conclusion where we summarize our findings and provide an outlook into the future.

Chapter 2

Technical Background

2.1 Introduction to Machine Learning

Machine Learning (ML) can be defined as a set of algorithms and statistical models that are used to efficiently solve tasks, relying on patterns rather than explicit instructions. The aim of Machine Learning is to teach computers how to "learn" and act, by leveraging algorithms that are able to infer patterns from data. According to Tom Mitchell, "*a computer program is said to learn from experience E with respect to some class of tasks $\mathcal{T}$ and performance measure P, if its performance at tasks in $\mathcal{T}$, measured by P, improves with experience E*"[42]. In practice, ML is the combination of powerful statistical models with compute in order to acquire knowledge from observations. The acquired knowledge can then be used in order to generate new data or take informed decisions based on observations.

Machine Learning enables us to address various tasks, $\mathcal{T}$, ranging from machine translation and object classification to anomaly detection. Most of these tasks, partially, describe problems that humans have solved, with varying efficiency, but are difficult for a computer to tackle. The difficulty often lies in the formal description of the problems; some of them are intuitively solved by humans, or the specification of the necessary knowledge to solve them. The tasks that can be solved by ML models also define the possible applications of ML in real-life scenarios such as recommendation systems, spam filtering or message autocomplete systems.

The performance measure, P, is an integral part of the construction and evaluation of ML models. Most measures are specific to particular tasks and will be more formally defined in section 2.3.1.

2.2 Learning Paradigms

Based on the experience E, Machine Learning paradigms can be divided into two broad categories: *unsupervised* and *supervised*. The main difference between the two categories is the kind of data that a Machine Learning algorithm is able to leverage during the learning process. The first category of algorithms, has access to input data only, while the second one has, additionally, access to desired outputs (or target data) for the corresponding input data. There are other variants such as *semi-supervised* learning algorithms where the provided data is a combination of the previous categories and *self-supervised* learning algorithms where the target data are automatically created from input data alone. Other categories also exist, such as reinforcement learning or weakly-supervised learning but are not employed in this work.

The data that a Machine Learning algorithm is able to *experience* during the learning

process is called a dataset and is composed of many examples, called data points. Each example is represented as a vector $\mathbf{x} \in \mathbb{R}^d$ of d features, where each feature contains the value of a specific attribute or measurement of the data. The examples are assumed to be drawn independently from the data-generating distribution $p_{data}(\mathbf{x})$[1]. A dataset can also be described as a design matrix $X \in \mathbb{R}^{n \times d}$, containing a different example in each of the n rows.

2.2.1 Unsupervised Learning

The aim of an unsupervised learning algorithm is to infer useful patterns regarding the structure of a dataset X. In many scenarios we want to model the true data-generating distribution $p_{data}(\mathbf{x})$[2] from the empirical distribution $\hat{p}_{data}(\mathbf{x})$, defined by the dataset. Besides modelling the data-generating distribution, such models can, additionally, be used in order to generate instances from the modeled distribution and are, in this case, called *generative* models. Another class of unsupervised learning algorithms is used for *clustering*. Clustering is the task of grouping data points together, forming clusters, in such a way that data points in the same cluster are similar, and different from data points, belonging to other clusters.

2.2.2 Supervised Learning

In the supervised learning paradigm, each example, $\mathbf{x}$, of the dataset is associated with a label or target, $\mathbf{y}$. The goal of the supervised learning algorithm is to predict the label $\mathbf{y}$ for each observation of the dataset. This process can also be seen as inferring a function $f : f(\mathbf{x}) = \mathbf{y},\ \mathbf{x} \in X,\ \mathbf{y} \in Y$ that maps an input sample to an output sample or as estimating the conditional probability $p(\mathbf{y}|\mathbf{x})$. The algorithm then uses the inferred function in order to map new examples. Supervised learning can be viewed as a process where a teacher or a "supervisor" instructs the algorithm to infer the correct mapping from X to Y. However, in practice, the only supervision is the labels provided to the algorithm as part of the dataset (labeled data).

Supervised problems can be further divided into two categories: *regression* and *classification*. The main difference between them is that the output variable in regression tasks is numerical (continuous) while in classification it is categorical (discrete). Depending on the setting, classification can be *binary*, *multi-class* or *multi-label*. Binary classification deals with only two classes, multi-class with more than two and multi-label is a multi-class problem where each input sample can be associated with more than one label.

2.2.3 Other variants of learning

Unsupervised and supervised learning are not formally distinguishable and the lines between them are often blurred. As a result, learning techniques such as *self-supervised learning* and *semi-supervised learning* cannot be easily identified as belonging to one of the two aforementioned categories.

Self-supervised learning is a technique in which labels are automatically generated from unlabeled data. However, there is not a clear distinction from the previous categories since labels do exist but are not an identifiable part of the original dataset. The labels in self-supervised learning do not require human labour but are inferred by leveraging relations between different input signals.

[1] We assume that the examples of a dataset are independent and identically distributed, i.e. i.i.d.

[2] We will use the notation $p(\mathbf{x})$ to refer to the data-generating distribution $p_{data}(\mathbf{x})$, too.

Semi-supervised learning is an approach to ML that combines a small amount of labeled data with a large amount of unlabeled data. The unlabeled data is employed in order to improve the performance of a learning algorithm, based on specific assumptions regarding the data-generating distribution $p_{data}(\mathbf{x})$[3]. Both variants exploit the abundance of unlabeled data to augment the initial dataset and achieve improved performance on downstream tasks.

2.3 Learning Process

As we explained in the previous section, a machine learning algorithm is equipped with a set of examples, which is called dataset or *training set*. During the learning process, otherwise called *training*[4], the model leverages the training set in order to appropriately modify its parameters according to some performance measure, P, evaluated in the training set, called training error. This is similar to a classic optimization problem.

The main challenge in machine learning, however, is that we additionally require our algorithms to perform well on previously unseen data. The ability of an algorithm or a model to adapt to new examples is called *generalization* and will determine its success on real world applications. In order to generalize, machine learning algorithms often make assumptions regarding the functions that should be learned. A widely used assumption for the function of the input that is learned is the *smoothness prior* or smoothness property[5]. We measure the generalization by applying the same performance measure, P, on a holdout set of examples, called *test set*. The error on the test set, namely the test error or generalization error or *risk*, is defined as the expected error of a model on new inputs.

Contrary to optimization, in machine learning, we do not have access to the true data-generating distribution $p_{data}(\mathbf{x}, \mathbf{y})$[6], but only the empirical distribution $\hat{p}_{data}(\mathbf{x}, \mathbf{y})$. Instead of optimizing the risk directly, we, therefore, optimize the training error and hope that the risk will also decrease, a process called *empirical risk minimization*. Although, the test set is different from the training set we make the assumption that the two datasets are identically distributed, drawn from the distribution $p_{data}(\mathbf{x}, \mathbf{y})$. This allows us to use the training error as a an indication of the test error and indirectly achieve our goal of generalization.

In order to measure generalization during the learning process, we further split the dataset into training set and *validation set*. The training set is then used to learn the model's parameters while the validation set is used to estimate the generalization error during training, via the *validation error*. In addition, machine learning models often contain parameters that cannot be learned during training, namely *hyperparameters*, that are also decided based on the model's performance on the validation set.

If a machine learning model has not significantly decreased the error, after the learning process, we say that the model underfits the data. In the case of a considerable gap between the training and the validation error we say that the model overfits the training data. Overfitting and underfitting are two main challenges in ML and can be addressed via regularization and other architectural choices that will be presented in subsequent sections.

[3] Assumptions in semi-supervised learning are: the continuity assumption (points that are close to each other are more likely have the same label), the cluster assumption (data can be divided into clusters where each point in the cluster has the same label) and the manifold assumption (data lie approximately on a manifold of much lower dimension than the input space).

[4] We also say that a machine learning model "fits" the training data.

[5] The smoothness prior assumes that the function which should be learned satisfies the condition $f(x) = f(x + \epsilon)$.

[6] In the unsupervised learning setting we are only concerned about $p_{data}(\mathbf{x})$.

2.3.1 Loss functions

In the previous section, the notion of a performance measure, P, was introduced, as a means to evaluate machine learning models. In most machine learning scenarios, the performance measure that we want to maximize or minimize cannot be optimized efficiently. We therefore introduce a *loss function*, $\mathcal{L}$, that quantifies the error related to the particular performance measure. Formally, a loss function maps values of one or more variables to a real number according to a distance measure between those values. We assume that by minimizing this surrogate loss function we will also optimize P indirectly.

The loss function that we want to minimize can be defined as the expectation of the per instance loss function L, over all possible data:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(x,y)\sim p_{data}} L(f(\mathbf{x};\boldsymbol{\theta}), \mathbf{y}), \tag{2.1}$$

where $f(\mathbf{x};\boldsymbol{\theta})$ is the inferred mapping of our model and $\boldsymbol{\theta}$ are the model parameters. In the empirical risk minimization setting, however, we can only minimize the loss on the training set:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(x,y)\sim \hat{p}_{data}} L(f(\mathbf{x};\boldsymbol{\theta}), \mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N} L(f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{y}_i)^{7}, \tag{2.2}$$

where $f(\mathbf{x}_i;\boldsymbol{\theta})$ is the predicted output of the model when the input is $\mathbf{x}_i$, $\mathbf{y}_i$ is the corresponding label and N the number of examples in the training set.

We will now present the loss functions that are employed in this work. Since both supervised and unsupervised algorithms can bee seen as estimating a probability function, we consider useful to first introduce the concept of *entropy*. Entropy, or Shannon entropy, of a distribution, is a measure of the expected amount of information in an event drawn from a distribution[8]. Entropy can also be defined as the average number of bits required to encode the information contained in a random variable:

$$H(p) = \mathbb{E}_{x\sim p}[-\log p(x)] = -\sum_x p(x)\log p(x) = \sum_x p(x)\log\frac{1}{p(x)} \tag{2.3}$$

Entropy effectively quantifies the the amount of uncertainty of a probability distribution. Probability distributions that are nearly deterministic will have low entropy, while distributions that are closer to uniform will have high entropy.

In order to compare two distributions we use *cross entropy*. Cross entropy is a measure of distance between two probability distributions for a given random variable or set of events. Cross entropy can also be defined as the average number of bits needed to identify an event drawn from the set if the coding scheme used for the set is optimized for an estimated probability distribution q, rather than the true distribution p. Formally:

$$H(p,q) = \mathbb{E}_{x\sim p}[-\log q(x)] = -\sum_x p(x)\log q(x) = \sum_x p(x)\log\frac{1}{q(x)} \tag{2.4}$$

Cross entropy is commonly used as a loss function in machine learning. In the classification setting, the first distribution refers to the empirical conditional distribution

[7] $\hat{p}_{data}$ is the empirical distribution of the i.i.d dataset examples.
[8] This is a definition of entropy from the Information Theory perspective.

$\hat{p}_{data}(\mathbf{y}|\mathbf{x})$ and the second distribution is the estimation of the true probability by our model, $p_{model}(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$. In this case the equation 2.4 can be written as:

$$H(\hat{p}_{data}, p_{model}; \mathbf{x}) = -\sum_{i=1}^{C} \hat{p}_{data}(y = i|\mathbf{x}) \log p_{model}(y = i|\mathbf{x};\boldsymbol{\theta}), \tag{2.5}$$

where C is the number of classes for the particular classification problem. In such problems, each label $\mathbf{y}$ represents the true multinomial distribution[9] over the categories $1, 2, \ldots, C$, $y_i = \hat{p}_{data}(y = i|\mathbf{x})$. For each input, the model produces a corresponding output vector $\hat{\mathbf{y}}$, $\hat{y}_i = p_{model}(y = i|\mathbf{x};\boldsymbol{\theta})$. The per-instance cross entropy loss can thus be simplified as:

$$L_{cross-entropy}(\mathbf{y}, \hat{\mathbf{y}}) = H(\hat{p}_{data}, p_{model}; \mathbf{x}) = -\sum_{i=1}^{C} y_i \log \hat{y}_i \tag{2.6}$$

The loss function of the equation 2.6 is also called *categorical cross entropy loss*. When $C = 2$, the problem is reduced to a binary classification problem: $y_i \in \{0, 1\}$ and $p(y_i = 1|\mathbf{x}) = 1 - p(y_i = 0|\mathbf{x})$. In this case, the *binary cross entropy loss* can be calculated as:

$$L_{binary\ cross-entropy}(\mathbf{y}, \hat{\mathbf{y}}) = -y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i) \tag{2.7}$$

The probability of the ith class is y_i, while the empirical probability of class i is $\hat{y}_i$, thus, the multinomial distribution for each sample takes the form: $p_{model}(y = i|\mathbf{x};\boldsymbol{\theta}) = \prod_{i=1}^{C} \hat{y}_i^{y_i}$. The log-likelihood of each sample then becomes:

$$\log p_{model}(y = i|\mathbf{x};\boldsymbol{\theta}) = \sum_{i=1}^{C} \log \hat{y}_i^{y_i} = \sum_{i=1}^{C} y_i \log \hat{y}_i \tag{2.8}$$

We observe from equations 2.6 and 2.8 that for a classification problem, cross entropy is equivalent to the negative of log-likelihood.

2.3.2 Optimization

In order to achieve low training error and, consequently, generalization, the loss functions, $\mathcal{L}$, that were introduced in the previous section, must be minimized. A common solution to this minimization problem, in the context of machine learning and deep learning, is the *gradient descent* algorithm[10]. Gradient descent minimizes a function by iteratively moving in the direction of the steepest descent, defined by the negative of the gradient. We, therefore, iteratively update the model parameters, $\boldsymbol{\theta}$, in the opposite direction of $\nabla_{\boldsymbol{\theta}}\mathcal{L}$:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \tag{2.9}$$

where η is the *learning rate* that determines the magnitude of each update. The gradient of equation 2.2 is:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{(x,y)\sim\hat{p}_{data}} L(f(\mathbf{x};\boldsymbol{\theta}), \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}_i;\boldsymbol{\theta}), \mathbf{y}_i) \tag{2.10}$$

[9] In fact, this distribution is a special case of the multinomial distribution for a single trial, called categorical distribution or multinoulli distribution.

[10] Gradient Descent belongs to class of optimization algorithms that make use of the derivative in order to optimize a function, called first-order optimization algorithms. Other optimization algorithms involve the use of the second derivative (second-order optimization algorithms) or no derivative at all (derivative-free algorithms).

This is also known as batch gradient descent, due to the fact that it processes all examples in a single, large batch. However, the gradient of equation 2.10 is computationally expensive, since it has to be computed over the entire dataset, for each update. In order to avoid such heavy computation and motivated by the possible redundancy in the dataset, we randomly sample a number of examples from the dataset to compute the gradient. This technique is known as minibatch or *stochastic gradient descent*, due to the stochasticity that sampling introduces to the gradient descent algorithm:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}_i; \boldsymbol{\theta}), \mathbf{y}_i), \tag{2.11}$$

where m is the size of the minibatch. Increasing m results in more accurate estimates of the gradient but increases the computation required for each update. In order to make unbiased estimates of the gradient and remove possible biases in the order of examples in datasets, we also shuffle the examples before splitting the dataset into minibatches.

The optimization algorithms usually stop when a stopping criterion is met or when the algorithm arrives at the global minimum. However, the non-linearity of many machine learning models (especially neural networks) causes most loss functions to become nonconvex and therefore no global minimum exists. In this case, the training stops when the loss function has a very low value, has reached a local minimum or due to other convergence criteria.

2.4 Introduction to Deep Learning

Deep Learning (DL) is an approach of Artificial Intelligence (AI) that allows computers to learn form experience, E and acquire their own knowledge in terms of a hierarchy of concepts, with each concept defined through its relation to simpler ones [42].

Traditional machine learning models relied heavily on the representation of the data, namely features, in order to achieve superior performance in many tasks. The selection and engineering of the right features was not an easy task and often required human labour and domain knowledge about the particular problem. Deep learning, however, not only infers a mapping from the representation of the input data to the output data, but also learns the representation itself. This technique is called *representation learning* and in deep learning it is performed by building representations hierarchically, learning complex representations out of simpler ones.

The use of deep learning has led to numerous astonishing achievements of AI, such as beating two Jeopardy's world champions [34] in 2011, creating a network that recognizes high-level objects such as cats and human faces [63], while improving the state-of-the-art system in recognizing object categories from ImageNet pictures [99] by 70%, in 2012 and using convolutional neural networks to achieve state-of-the-art performance in ImageNet [58], in 2012. Another milestone of deep learning is AlphaGo [107], a reinforcement learning agent that beat the the world No.1 ranked Go player at the time, in 2017.

2.4.1 Artificial Neural Networks

Deep learning can be expressed as the combination of *artificial neural networks* (ANNs) with representation learning in order to efficiently solve problems. Artificial neural networks are biologically-inspired computing systems that learn to perform tasks without explicit programming. The term neural stems from the fact that the original learning systems were modeled after biological neural systems [73]. Neural systems still remain the

initial inspiration and a proof that intelligent behaviour is possible. However, deep learning has evolved and diverged from attempting to simulate the brain, to solving engineering applications with the use of large amounts of data and complex architectures. The quest of understanding how brain works by reverse-engineering it is still a field of study, different from deep learning, named *computational neuroscience*.

Although deep learning systems are not accurate models of neural systems it is helpful to introduce the building blocks of ANNs in comparison to the biological systems that motivated them. The central idea in both systems is that a large number of simple computational units can result to intelligent behaviour when used together. The basic computational unit of the brain and ANNs is a *neuron*. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately 10^{14} - 10^{15} edges, called synapses, while the biggest deep learning models to date have $2-17$ billion parameters (equivalent to synapses). Figure 2.1 compares a biological (a) and an artificial neuron (b). Both neurons receive input from different synapses and weigh these inputs by multiplying them according to learnable *weights*. In the biological neuron, the multiplication occurs in the dendrites and the weights are defined by the synaptic strength [54]. In both cases, the weights control the importance of each input from the other neurons. The multiplied inputs are then summed and if the sum is above a threshold, the biological neuron fires. On the contrary, an artificial neuron applies a non-linear, differentiable function, ϕ, to the weighted sum, after adding a bias term[11], and outputs a continuous number, y_κ. In both scenarios, the output of each neuron, is provided as an output to other neurons:

$$y_\kappa = \phi\left(w_{\kappa 0} + \sum_{i=1}^{m} w_{\kappa i} x_i\right) \tag{2.12}$$

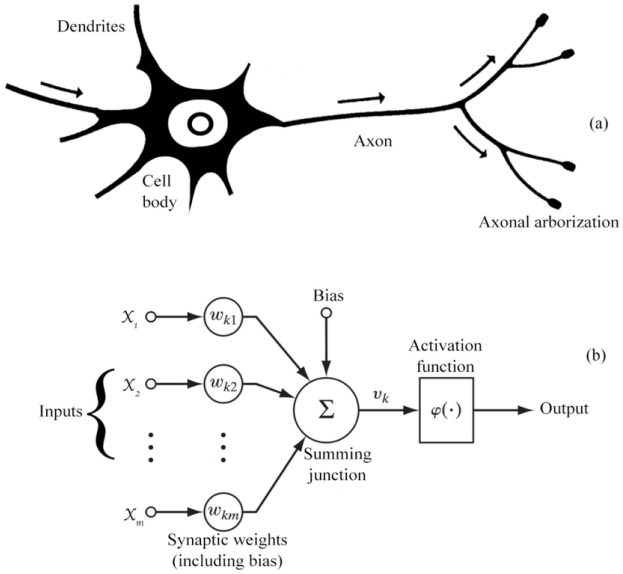


Figure 2.1: Similarity between a biological (top) and an artificial (bottom) neuron. Figure from [1].

Artificial neural networks are collections of neurons that are connected in order to estimate complex functions. Artificial neurons are organized in *layers*, composed of neurons

[11] The weights and bias are the parameters of the artificial neuron.

working in parallel, as depicted in Figure 2.2. Neurons of each layer receive input signals from the immediately preceding layer and provide output signals to the immediately following layer, forming *feedforward neural networks*. There are other type of neural networks that allow connections between neurons in the same or previous layers, known as *recurrent networks*. If each neuron from one layer is connected to all neurons from the following layer, we say that the layer is *fully connected* or *dense layer*.

The first layer that receives the input data is the *input layer* while the last layer that produces the result is the *output layer*. In between there are other layers that do not interact with the input or output data, but only with the intermediate representations, called *hidden layers*. The number of the hidden layers determines the *depth* of a neural network.

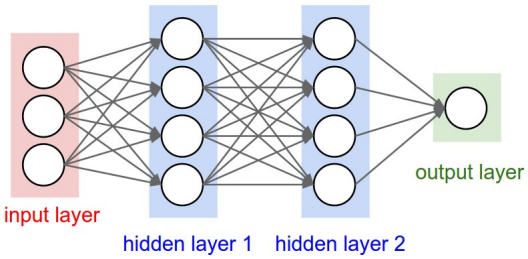


Figure 2.2: A feedforward neural network with 2 hidden layers. Figure from [54].

2.4.2 Activation Functions

The non-linear functions that are applied to the inputs of every neuron in neural networks are called *activation functions*. The non-linear nature of activation functions is essential to approximate non-linear functions, since linear activation functions would result in linear models, mapping input to output via matrix multiplication alone. Specifically the universal approximation theorem [24] states that a feedforward network with a linear output activation function and at least one hidden layer with any "squashing"[12] activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units [42]. Activation functions must also be differentiable, due to the gradient descent algorithm employed in neural networks optimization. This, also, implies that activation functions that saturate will produce very small gradients undermining the learning process. Some of the commonly used activation functions in deep learning are the following:

Sigmoid: The (logistic) sigmoid activation function is defined by:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.13}$$

The sigmoid function takes a real-valued number as input and "squashes" it between 0 and 1, as illustrated in Figure 2.3. It is commonly used to produce the p parameter of a Bernoulli distribution and is, thus, employed in the output layer in binary classification problems. The sigmoid saturates when its argument is very positive of very negative, where the output becomes almost flat and the gradient almost zero. In addition, the the sigmoid

[12] While this was originally proven for activation functions that saturate for both very positive and very negative values, it was later shown [66] that the class of deep neural networks is a universal approximator if and only if the activation function is not polynomial.

outputs are not zero-centered which can cause a problem in the learning dynamics of a neural network.

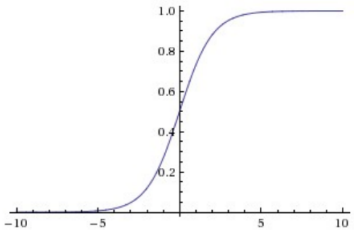


Figure 2.3: The sigmoid activation function.

Tanh: The tanh, or hyperbolic tangent, computes the following function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.14}$$

Tanh activation function is similar in shape to the sigmoid, but "squashes" the input between -1 and 1, as can be seen in Figure 2.4. In fact, the tanh is a scaled version of the sigmoid function, as from equations 2.13 and 2.14 it holds that $\tanh(x) = 2\sigma(2x) - 1$. However, unlike sigmoid, the outputs of the tanh function are zero-centered, which often makes tanh preferable over sigmoid, as a non-linear activation function.

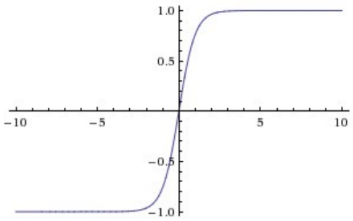


Figure 2.4: The tanh activation function.

Softmax: The softmax, or normalized exponential, activation function is formally given by:

$$softmax(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \ where \ \mathbf{x} = x_1, x_2, \ldots, x_K \tag{2.15}$$

The softmax function takes as input a vector of K real numbers and normalizes it into a probability distribution consisting of K probabilities, proportional to the exponential of the input numbers. It is commonly used to represent probability distributions over a discrete variable with K possible values, and is, therefore, employed in the output layer in mutliclass classification problems. The exponential in the equation 2.15 ensures that all components are positive and the denominator forces the sum of the components to be 1. As a result, the components can be interpreted as probabilities. In the case that an input component x_i is maximal and much greater than the other components, the softmax function saturates to 1 for the maximal component and to 0 for all the other components.

Rectified Linear Unit(ReLU): ReLU is one of the most commonly used activation functions in hidden layers of deep learning models and computes the function:

$$f(x) = max(0, x) \tag{2.16}$$

ReLU is a half-rectifying linear activation function, as depicted in Figure 2.5. It is biologically inspired, in the sense that neurons in the human brain are considered to be sparsely activated: during the function of the brain some neurons are inactive, while others output a signal proportional to their input [40]. The ReLU activation is easy to implement and optimize due to its similarity with the linear function. Furthermore, its derivatives remain large when the activation function is active, facilitating the learning process. ReLU activation functions can lead some neurons to "die" during optimization, functioning permanently in the zero-state, which is not necessarily undesirable, since it leads to sparse representations. ReLU outputs are also not zero-centered.

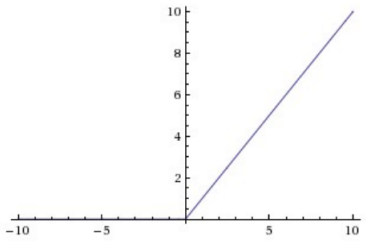


Figure 2.5: The ReLU activation function.

Gaussian Error Linear Unit(GELU): GELU is a recently introduced activation function that computes:

$$f(x) = 0.5x(1 + \tanh(\sqrt{2/\pi}(x + 0.044715x^3))) \tag{2.17}$$

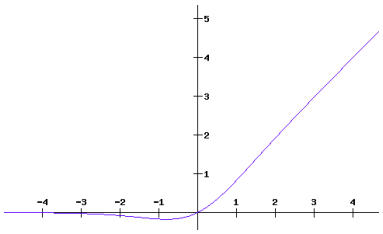


Figure 2.6: The GELU activation function.

As can be seen in Figure 2.6, GELU activation function is similar to ReLU apart from the nonlinearity close to zero. The GELU nonlinearity is the expected transformation of a stochastic regularizer which randomly applies the identity or zero map to a neuron's input [49]. It has been shown to improve performance in multiple tasks, and has been adopted in the Transformer architecture, that will be defined in subsequent sections.

2.4.3 Back Propagation

In section 2.3.2 we observed that in order to minimize the error we need to update the model parameters in the opposite direction of the gradient of the loss function. The analytic computation of the gradient for each parameter, individually, can be computationally expensive, motivating the need for more efficient algorithms. *Back-propagation* is a widely used algorithm in training neural networks that computes the gradient of the loss function with respect to each weight utilizing the chain rule of calculus [98].

Back-propagation computes the gradient one layer at a time, iterating from the output layer, allowing the algorithm to reuse information since many partial derivatives for

the chain rule have already been computed in previous steps. The term back-propagation comes in contradiction with forward propagation, which is used to describe the propagation of information in a neural network, from the input layer to the hidden layers and finally the output layer. The back-propagation algorithm can be used in almost any neural network architecture, including recurrent neural networks where a variant called *back-propagation through time* is employed. Modern deep learning frameworks utilize automatic differentiation techniques along with computational graphs in order to efficiently implement back-propagation.

2.4.4 Optimization Algorithms

In the previous section we introduced the algorithm that is responsible for computing the gradient, while in section 2.3.2 we presented the optimization algorithm that performs learning using the gradient, gradient descent and its most commonly used variant, stochastic gradient descent. Although stochastic gradient descent works satisfyingly well in neural network training, it has certain deficiencies. The learning rate of equation 2.9 needs to be carefully adjusted in order for the algorithm to converge within an acceptable number of steps. In addition, it is often needed to decrease the learning rate over time in order to ensure better convergence. Finally, in the equation 2.9 the same learning rate applies to all parameter updates, while the gradient of the loss function has not the same sensitivity across all parameters [42]. To counter these problems, other gradient-based optimization algorithms, also called *optimizers* have been introduced, some of which will be presented in this section. For the remaining section we will use $\mathbf{g}$[13] to refer to the approximation of the true gradient of the equation 2.11.

AdaGrad: AdaGrad algorithm [32], adapts the learning rate to each parameter, by scaling the learning rates inversely proportional to the square root of the sum of all the historical squared values of the gradient with respect to each parameter. This is accomplished, using a gradient accumulation parameter, $\mathbf{r}$, that is initialized to zero and is updated in every optimization step, using the computed gradient, $\mathbf{g}$:

$$\mathbf{r} = \mathbf{r} + \mathbf{g} \odot \mathbf{g} \tag{2.18}$$

The parameter update, then, becomes:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\eta}{\delta + \sqrt{\mathbf{r}}} \ , \tag{2.19}$$

where η is the global learning rate (with a default value of 0.01) and δ is a small constant, used for numerical stability. The update step of equation 2.19 leads to smaller updates for parameters with historically large partial derivatives and larger updates for parameters with smaller partial derivatives, over the past optimization steps [42]. However, the aggressive monotonic decrease of the learning rate, from the beginning of the optimization can result in suboptimal performance.

RMSProp: RMSProp algorithm (Hinton, 2012) improves the idea of gradient accumulation, by changing the sum of squared gradients into an exponentially decaying average:

$$\mathbf{r} = \rho\mathbf{r} + (1 - \rho)\mathbf{g} \odot \mathbf{g} \ , \tag{2.20}$$

where ρ is the decay rate, with a default value of 0.9. The parameter update is the same as in equation 2.19, with the suggested value of η being 0.001. The exponential decay of

[13]The gradient $\mathbf{g}$ is a vector, with each dimension associated with a parameter of the model.

RMSProp results in accumulation of gradients over the most recent past, which has been shown to be effective for deep neural networks.

Adam: Adaptive Moment Estimation (Adam) [56] is another adaptive learning rate optimization algorithm and has been widely adopted for the optimization of deep neural networks. The Adam algorithm employs the momentum of gradients, by accumulating an exponentially decaying moving average of gradients, in order to accelerate learning:

$$\mathbf{s} = \rho_1 \mathbf{s} + (1 - \rho_1)\mathbf{g} \ , \tag{2.21}$$

where $\mathbf{s}$ is again initialized to zero and ρ_1 is the decay rate with a default value of 0.9. According to equation 2.21, the more aligned the current gradient estimate, $\mathbf{g}$, is with the past gradients, $\mathbf{s}$,the more momentum it will gain. Adam, also, combines the momentum of equation 2.21 with the exponentially decaying average of the squared gradients of equation 2.20, with a decay factor, ρ_2 (with a default value of 0.999). These two factors can, additionally, be considered as the first-order moment (mean) and the second-order moment (variance) of the gradients. Due to their initialization as a vector of zeros the two moments can be, under certain circumstances and especially during initial optimization steps [56], biased towards zero and are therefore corrected:

$$\hat{\mathbf{s}} = \frac{\mathbf{s}}{1 - \rho_1^t} \ and \ \hat{\mathbf{r}} = \frac{\mathbf{r}}{1 - \rho_2^t} \tag{2.22}$$

The parameter update, finally, becomes:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\eta}{\delta + \sqrt{\hat{\mathbf{r}}}}\hat{\mathbf{s}} \tag{2.23}$$

In order to avoid large updates during early optimization steps which may be biased from initial training samples and can cause the model to deviate from a good solution, a *warmup* period has been proposed. During warmup steps the learning rate slowly increases from zero to a specified value, typically according to a specific schedule (e.g. linear), to stabilize learning and prevent divergence in early training. Warmup also enables adaptive algorithms such as Adam to acquire better estimations for the estimated moments. In current DL research, warmup (combined with the corresponding schedule) is employed for most optimizers, either for a specific number of steps or for a fraction of the total optimization steps.

2.4.5 Regularization

Generalization has been identified as a central challenge in machine learning. In section 2.3 we introduced the concept of overfitting, where the validation error is significantly larger than the training error, which can be an indication of large generalization error. In order to address such issues we employ *regularization*, which can be defined as "*any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error*" [42]. There are many generalization strategies: some put constraints on the model parameters, others add loss terms in the loss function and still others introduce modifications to the learning algorithm itself. Some of these regularization techniques, that are employed in this work, will be presented in this section.

L^2 Regularization: L^2 regularization is achieved by adding the square of the L_2 norm of the model parameters in the loss function with the intention of keeping the value of the parameters close to zero. The new loss function $\tilde{\mathcal{L}}(\boldsymbol{\theta})$, then, becomes:

$$\tilde{\mathcal{L}}(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \frac{\alpha}{2}\|\boldsymbol{\theta}\|_2^2 = \mathcal{L}(\boldsymbol{\theta}) + \frac{\alpha}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} \ , \tag{2.24}$$

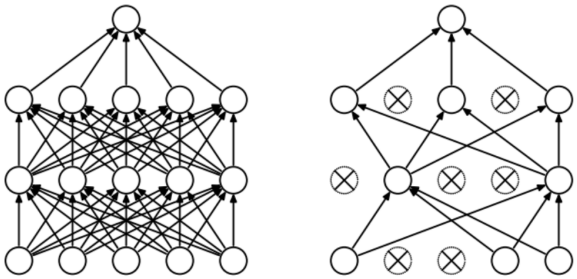
where α is the decay rate. The update step of equation 2.9, then, becomes:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}(\mathcal{L}(\boldsymbol{\theta}) + \frac{\alpha}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}) = (1 - \eta\alpha)\boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) \tag{2.25}$$

The additional loss term results in multiplicatively shrinking the parameter vector by a constant factor in each step, which is why this technique is also called *weight decay*. In this way large updates in the parameters, which may be a sign of overfitting can be avoided, favouring lower parameter values.

Early Stopping: Early stopping aims to prevent overfitting by determining the number of training steps. The goal of early stopping is to obtain the model with the lowest validation error, over training. Thus, every n training steps[14], the validation error is computed and if it improves, a copy of the current model parameters is stored. When the algorithm terminates, the model parameters are set to those with the lowest validation error, previously stored, rather than the latest. Early stopping can additionally modify the stopping criterion, by terminating the algorithm if the validation error has not improved over some pre-specified number of iterations. This regularization technique is very popular since it is easy to implement, does not require modification in the training procedure or loss function and can be combined with other regularization techniques.

Dropout: Dropout is an effective regularization technique [109] that prevents co-adaptation of neurons, in which a neuron is only helpful in the context of several other specific neurons, during training. The key idea of dropout is to randomly drop neurons (with probability p) along with their (outgoing) connections from a neural network during training, as seen in Figure 2.7. This is implemented by sampling and applying a different binary mask for each unit, at each iteration step. The masks are independently sampled and differ across iterations. Dropout can be viewed as training an ensemble of subnetworks that emerge by randomly removing units from a base network. A fraction of the possible subnetworks are actually trained (due to the exponential amount of possible configurations) and their parameters are shared. In addition, they are mostly trained for one iteration using only a small subset of the available data (minibatch). Although dropout was originally introduced for feedforward and convolutional neural networks it has been successfully applied to recurrent neural networks, as well [131] [35] and is now part of most deep learning architectures.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.7: A neural network without (Left) and with (Right) dropout. Figure from [109].

Batch Normalization: Batch Normalization (BN) [52] is an adaptive reparametrization method that addresses the problem of *covariate shift*. In the context of DL, covariate shift is framed as the change of a layer's inputs distribution due to the parameter change of the previous layers, during training. To limit covariate shift, the activations[15] of each layer

[14] This is usually after the model has iterated over the entire dataset once, called an *epoch*.

[15] We note that the activations of a layer are the inputs of the next layer.

are normalized (zero mean and unit variance) over each minibatch. This allows each layer to learn on a more stable distribution of inputs, and can accelerate the training process by enabling the use of higher learning rates. This normalization, however, could hurt the model's expressivity. Therefore, in practice, BN allows the network to learn parameters γ and β that can convert the mean and variance to any value.

Layer Normalization: Layer Normalization (LN) [3] is another normalization method that normalizes the input features across the feature dimension, in contrast to BN that normalizes the activations across the batch dimension. This allows for more parallelization since the statistics (mean and variance) can be calculated independently for each example. In addition, contrary to BN, the effect of this method does not depend on the batch size.

2.5 Deep Neural Networks

A crucial factor of the success of deep learning models is the variety of architectures that are designed for a various classes of problems. In this section we will present some of the most common deep neural network architectures that are also employed in this work.

2.5.1 Challenges in Deep Neural Network Training

Although deep neural networks have achieved remarkable results in many tasks, training such networks poses certain challenges that must be addressed to facilitate the learning process. Most of these challenges are associated with the optimization of deep neural networks via the back-propagation algorithm and are thus present, to some extent, in every deep learning architecture

During the training process of deep neural networks with back-propagation, extremely large values of gradient can occur (they can go to infinity) which result in large parameter updates, causing instability (even overflow) to the network. On the other hand, gradients can also have extremely small values (almost zero), leading to infinite-small updates, which hinder the learning process. These two cases have been identified in neural network training [8], [50] as the *vanishing gradients* and the *exploding gradients* problems.

In order to overcome the exploding gradients problem, methods such as weight regularization (described in section 2.4.5) and *gradient clipping* have been proposed. Gradient clipping avoids extremely large parameter updates, by clipping the gradient when it exceeds a specific threshold, during training[16]. The vanishing gradients problem can be addressed via activation functions that saturate on one direction only (such as ReLU and GELU from section 2.4.2) and *residual networks*. Residual networks (ResNets) [47] are networks that contain *skip connections* or shortcuts that bypass some layers in order to connect layers further behind the preceding layer to a given layer, as depicted in Figure 2.8. This allows gradients to propagate faster to deep neural networks before they can be reduced to extremely small values.

2.5.2 Feedforward Neural Networks

Deep feedforward networks, or feedforward neural networks (FFNNs), or multilayer perceptrons (MLPs) are feedforward neural networks (defined in section 2.4.1) with multiple layers between the input and the output layer. The difference between a simple feedfor-

[16]There are various implementations of gradient clipping: one suggestion is to clip the parameter gradient from a minibatch element-wise [117] and another is to clip the norm of the gradient [85].

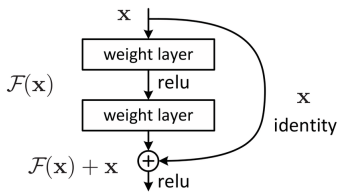


Figure 2.8: A residual network block. Figure from [47].

ward neural network is, essentially, the number of hidden layers, that also determines the depth of the FFNN, as shown in Figure 2.9.

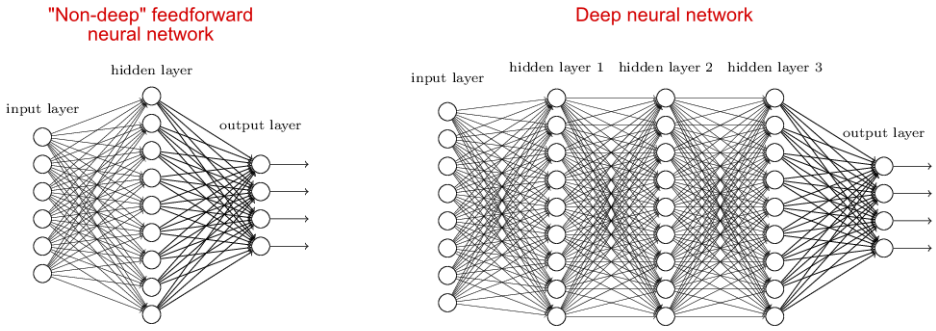


Figure 2.9: Feedforward neural network (Left) vs deep feedforward neural network (Right). Figure from [82].

2.5.3 Recurrent Neural Networks

Recurrent neural networks, or RNNs, are a family of networks, ideal for processing sequential data due to their memory structure and reccurency [98]. RNNs address the limited capabilities of FFNNs for sequence processing that require a constant, pre-specified sequence length and use separate parameters for each sequence point. In order to efficiently process sequences, RNNs share parameters across sequence points, which also enables them to generalize across variable sequence lengths. Thus, at each point in the sequence, the output of a recurrent neural network is a function of the previous outputs and the current input. In order to retain information of previous steps (outputs) in the sequence, RNNs maintain an internal state that is updated at each step, forming a kind of memory. Every RNN variant, computes the following formula at each point in the sequence, also called time step[17], t:

$$\mathbf{h_t} = f(\mathbf{h_{t-1}}, \mathbf{x_t}, \boldsymbol{\theta}) \ , \tag{2.26}$$

where $\mathbf{h_t}$ is the state of the RNN, also called *hidden state*, at time step t, $\mathbf{x_t}$ is the input at time step t, $\boldsymbol{\theta}$ the model parameters and f the particular function of the model.

Recurrent neural networks consist of a chain of repeating modules, also called cells. The structure of the module defines the different recurrent neural network variants. Each chain of repeating modules is called an RNN layer. Similar, to FFNNs the output of an RNN layer can be provided as input to another RNN layer. This process of *stacking*

[17] We often use the term time step to refer to different points in a sequence, although time order between sequence points does not necessarily exist.

RNN layers increases the network depth and creates new representations at higher levels of abstraction.

RNN can be, additionally, viewed as a cyclic graph which, when unrolled in time, resembles a FFNN with a dynamic number of hidden layers that receives an input at each "layer". In order to train RNNs, the back-propagation algorithm is applied to the unrolled computational graph, which is in this case called *back-propagation through time*.

Vanilla RNN: Figure 2.10, illustrates the simplest from of RNN, the vanilla RNN. It takes as input a sequence of t feature vectors $\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_t}$, computes a sequence of t hidden states $\mathbf{h_0}, \mathbf{h_1}, \ldots, \mathbf{h_t}$ and outputs a sequence of t output vectors $\mathbf{o_0}, \mathbf{o_1}, \ldots, \mathbf{o_t}$. At every time step, RNN computes the following functions:

$$\mathbf{h_t} = \tanh(\mathbf{W_{hh}}\mathbf{h_{t-1}} + \mathbf{W_{xh}}\mathbf{x_t} + \mathbf{b_h}) \tag{2.27}$$

$$\mathbf{o_t} = \mathbf{W_{hy}}\mathbf{h_t} + \mathbf{b_y} \ , \tag{2.28}$$

where $\mathbf{o_t}$ is the output vector at time step t, $\mathbf{b_h}$ the bias for the hidden state and $\mathbf{b_y}$ the bias for the output. The RNN also contains three weight matrices: the input-to-hidden matrix, $\mathbf{W_{xh}}$, the hidden-to-hidden matrix, $\mathbf{W_{hh}}$ and the hidden to output matrix, $\mathbf{W_{hy}}$.

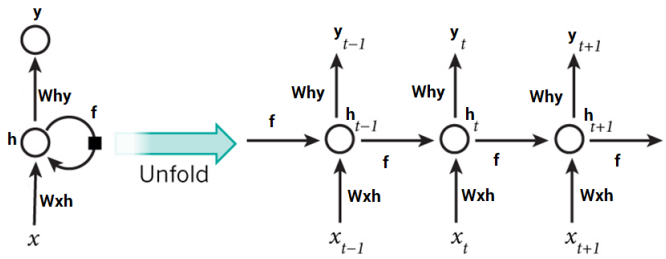


Figure 2.10: RNN as a cyclic graph and the equivalent unrolled version. Figure from [64].

Theoretically, even the simplest form of RNNs, the vanilla RNN, has infinite memory capacity, which means that it can process sequences of infinite length. In practice, however, this is not exactly the case. Exploding and vanishing gradients are a common problem of RNNs due to recurrency, limiting the effective maximum sequence length.

LSTM: Long Short-Term Memory Networks (LSTMs) [50] are a special kind of RNNs that are capable of handling long-term dependencies. LSTMs leverage gating mechanisms in order to retain or remove information over a sequence, which enables them to process longer sequences. The LSTM extends the idea of memory structure of RNNs by adding a *cell state*, in addition to the hidden state, that can be also modified, through the same gating mechanisms, as illustrated in Figure 2.11. Given an input sequence of t feature vectors $\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_t}$, the LSTM computes the following functions at each time step:

$$\mathbf{f_t} = \sigma(\mathbf{W_f}\mathbf{x_t} + \mathbf{U_f}\mathbf{h_{t-1}} + \mathbf{b_f}) \tag{2.29}$$

$$\mathbf{i_t} = \sigma(\mathbf{W_i}\mathbf{x_t} + \mathbf{U_i}\mathbf{h_{t-1}} + \mathbf{b_i}) \tag{2.30}$$

$$\mathbf{o_t} = \sigma(\mathbf{W_o}\mathbf{x_t} + \mathbf{U_o}\mathbf{h_{t-1}} + \mathbf{b_o}) \tag{2.31}$$

$$\tilde{\mathbf{C}}_\mathbf{t} = \tanh(\mathbf{W_C}\mathbf{x_t} + \mathbf{U_C}\mathbf{h_{t-1}} + \mathbf{b_C})^{18} \tag{2.32}$$

$$\mathbf{C_t} = \mathbf{f_t} \odot \mathbf{C_t} + \mathbf{i_t} \odot \tilde{\mathbf{C}}_\mathbf{t} \tag{2.33}$$

$$\mathbf{h_t} = \mathbf{o_t} \odot \tanh(\mathbf{C_t}) \tag{2.34}$$

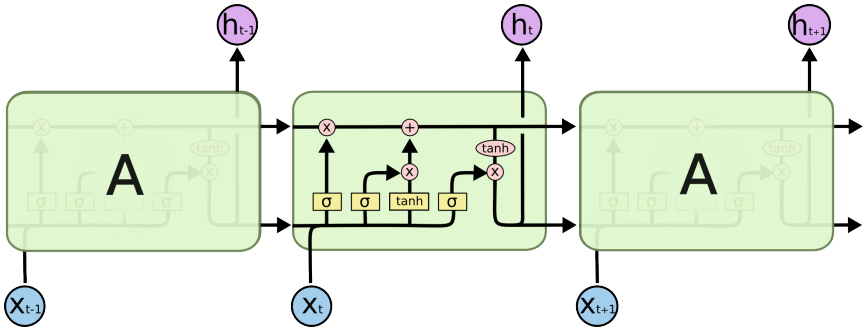


Figure 2.11: The repeating LSTM module. Figure from `Understanding LSTMs - colah's blog`.

Forget gate ($\mathbf{f_t}$): The forget gate takes as input the previous hidden state, $\mathbf{h_{t-1}}$, and the current input vector, $\mathbf{x_t}$, and outputs a number between 0 and 1 (using the sigmoid function) for every dimension of the cell state, $\mathbf{C_t}$. The function of the forget gate can be seen as "deciding" which part of the previously stored information will be removed in favour of new information, provided in the sequence.
Input gate ($\mathbf{i_t}$): The input gate, similar to forget gate, leverages the previous hidden state, the current input vector, and the sigmoid function, to output a number between 0 and 1 for every dimension of the cell state. The function of the input gate is to "select" which part of the input must be stored in the LSTM memory.
Output gate ($\mathbf{i_t}$): The output gate, similar to the previous gates, "decides" which part of the cell state will be transferred in the hidden state (and consequently the output), based on the current input and the previous hidden state.
The information of the current time step that will be, potentially, stored in the cell state, namely the **candidate cell state**, $\tilde{\mathbf{C}}_\mathbf{t}$, is selected with the same mechanism as the LSTM gates. The main difference is the non-linearity that is applied to the input, which in this case is the tanh activation function. In order to, finally, update the cell state, parts of the previous cell state are discarded via the forget gate, $\mathbf{f_t} \odot \mathbf{C_t}$, and new candidate values of the cell state are integrated via the input gate, $\mathbf{i_t} \odot \tilde{\mathbf{C}}_\mathbf{t}$. These two factors are then summed to form the updated cell state, $\mathbf{C_t}$. The LSTM outputs a hidden state $\mathbf{h_t}$, by applying the *tanh* activation function to the updated cell state, which is then multiplied with the output gate, $\mathbf{o_t} \odot \tanh(\mathbf{C_t})$. The new hidden state and the new cell state are fed back to the module for the next time step.

A variant of the classic RNN architecture is the *bidirectional* recurrent neural network [103], which consists of two RNNs that run in parallel: one on the input sequence and the other on the reverse of the input sequence. At each time step, the hidden state of the bidirectional RNN is the concatenation of the forward and backward hidden states $h_t = \overrightarrow{h_t} || \overleftarrow{h}_t$. This setup allows the hidden state to capture both past and future information.

Another extension of the classic RNN architecture is the *sequence to sequence* (seq2seq) architecture [18], [113], also called Encoder-Decoder architecture. The seq2seq architecture employs two RNNs in order to transform a variable-legnth input sequence to another variable-length output sequence, as shown in Figure 2.12. The first RNN, called the *encoder* maps the input sequence to a fixed vector, also called *context vector*, while the second RNN, the *decoder*, generates the output sequence, conditioned on the context vector. A simple

[18] An equivalent form for the equations of the LSTM is: $\mathbf{f_t} = \sigma(\mathbf{W_f}[\mathbf{x_t}; \mathbf{h_{t-1}}] + \mathbf{b_f})$, $\mathbf{i_t} = \sigma(\mathbf{W_i}[\mathbf{x_t}; \mathbf{h_{t-1}}] + \mathbf{b_i})$, $\mathbf{o_t} = \sigma(\mathbf{W_o}[\mathbf{x_t}; \mathbf{h_{t-1}}] + \mathbf{b_o})$, $\tilde{\mathbf{C}}_\mathbf{t} = \tanh(\mathbf{W_C}[\mathbf{x_t}; \mathbf{h_{t-1}}] + \mathbf{b_C})$, where $\mathbf{W} = [\mathbf{W}; \mathbf{U}]^\top$.

form of conditioning the decoder on the encoder is to use the last hidden state of the encoder as the initial hidden state of the decoder [113] (Figure 2.12) or as an extra input to each decoder hidden state [18].

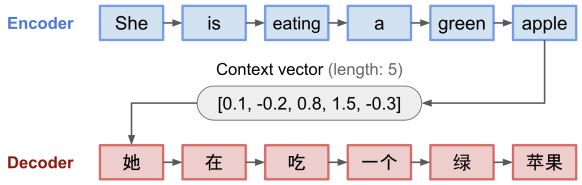


Figure 2.12: Sequence to sequence architecture for translating a sentence from English to Chinese. Figure from Weng [123].

2.5.4 Attention Mechanisms

A deficiency of the original sequence to sequence architecture is the fact that the context vector acts as a bottleneck of information, since it is a compression of the whole input sequence. To mitigate this effect, Bahdanau, Cho, and Bengio [4] proposed *attention*, a mechanism that allows the decoder of a seq2seq model to search the input sequence to find information that is relevant to the current step of the output sequence. The decoder is, thus, able to attend to specific parts of the input that are useful for a particular task, a concept similar to human attention. Although it was originally introduced for sequence to sequence architectures, various attention mechanisms have been applied to many neural network architectures.

Given a sequence of vectors $\mathbf{h_0}, \mathbf{h_1}, \ldots, \mathbf{h_n}$, and a vector $\mathbf{s_i}$, also called the *query vector*, which may or may not be part of the sequence, the attention mechanism produces a representation of the sequence based on the alignment with the query vector. In the case $\mathbf{s_i}$ is part of the sequence, the attention mechanism is called *self-attention* or *intra-attention*. The attention mechanism utilizes an *alignment score function* to compute the alignment scores (similarities) between a pair of vectors, also called attention energy, $e_{i,j} = align(\mathbf{s_i}, \mathbf{h_j})$. The scores are then passed through a softmax function, $\alpha_{i,j} = softmax([e_{i,1}, e_{i,2}, \ldots, e_{i,n}]^\top)_j$ and can be viewed as a distribution over the sequence vectors. Finally, the context vector is formed as a weighted sum of the sequence vectors, according to the normalized scores, $\mathbf{c_i} = \sum_{j=1}^{n} a_{i,j}\mathbf{h_j}$. We will now present different alignment functions and variants of the attention mechanism, that are employed in this work.

Additive attention: The original attention mechanism, as introduced in [4], employed an MLP with a tanh activation function to produce the alignment scores:

$$e_{i,j} = \mathbf{v_a}^\top \tanh(\mathbf{W_a}\boldsymbol{s_i} + \mathbf{U_a}\boldsymbol{h_j})^{19} \ , \tag{2.35}$$

where $\mathbf{v_a}$, $\mathbf{W_a}$, and $\mathbf{U_a}$ are learned parameters of the attention mechanism. The vector $\mathbf{v_a}$ can be viewed as a high level representation of a "fixed query", that focuses on specific parts of the sequence [67], [128]. In the self-attention scenario it is common to omit the query vector $\mathbf{s_i}$ in additive attention : $e_j = \mathbf{v_a}^\top \tanh(\mathbf{W_a}\boldsymbol{h_j})$.

Multiplicative attention: An alternative alignment score function [71] is implemented with the use of a single matrix:

$$e_{i,j} = \mathbf{s_i}^\top \mathbf{W_a} \mathbf{h_j} \ , \tag{2.36}$$

[19] An equivalent form of this equation is: $e_{i,j} = \mathbf{v_a}^\top \tanh(\mathbf{W_a}[\mathbf{s_i}; \mathbf{h_j}])$.

where $\mathbf{W_a}$ is again a learned matrix. If we set $\mathbf{W_a} = \mathbf{I}$ in the above equation, the alignment function becomes the dot-product, without any learnable parameters, $e_{i,j} = \mathbf{s_i}^\top \mathbf{h_j}$. This simple form of attention requires the query and the sequence vectors to be of equal size.

Query-Key-Value attention: Query-Key-Value attention can be viewed as a variant of the dot-product attention, described above. A simpler form of this attention mechanism was employed in the earlier versions of Memory Networks [110] but has been popularized by the Transformer architecture [118]. This attention mechanism is based on projecting all vectors to query, key and value vectors, via linear transformations:

$$\mathbf{q_i} = \mathbf{W^Q h_i} \ , \ \mathbf{k_i} = \mathbf{W^K h_i} \ , \mathbf{v_i} = \mathbf{W^V h_i}, \tag{2.37}$$

where $\mathbf{W^Q}$, $\mathbf{W^K}$, and $\mathbf{W^V}$ are learned matrices. Then, the alignments between a query vector and the key vectors are calculated, using the dot-product alignment function. Therefore, the matrices $\mathbf{W^Q}$ and $\mathbf{W^K}$ must have the same dimensions. In the Transformer architecture [118], the dot-product is additionally scaled by the square root of the dimension size of the query and key vectors, $\sqrt{d_k}$ [20]:

$$e_{i,j} = \frac{\mathbf{q_i}^\top \mathbf{k_j}}{\sqrt{d_k}} \tag{2.38}$$

Finally, the context vector is formed as a weighted sum of the value vectors, based on the normalized scores, $\mathbf{c_i} = \sum_{j=1}^{n} a_{i,j} \mathbf{v_j}$. The use of the matrix $\mathbf{W^Q}$ to project the original query vector, $\mathbf{s_i}$ to a transformed query vector $\mathbf{q_i}$ is essential for the self-attention scenario. Taking the dot-product of a vector with itself results in the magnitude of the vector squared which would strongly bias the attention mechanism towards attending only to the original query vector $\mathbf{s_i}$.

The attention mechanisms described above can be further refined to implement more than one attention functions. This is often referred to as *multiple attention hops* [110]. In the case of multiplicative attention, multiple hops can be implemented, by simply introducing multiple vectors, extending $\mathbf{v_a}$ to a matrix $\mathbf{V_a}$ [67]. For the Query-Key Value attention, multiple hops can be achieved with the use of multiple query, key and value projection matrices, $\mathbf{W_i^Q}$, $\mathbf{W_i^K}$, $\mathbf{W_i^V}$. The multiple hops in this case are also called *attention heads* [118].

2.5.5 Transformers

The attention mechanism has greatly enhanced many neural network architectures. One of the models that has benefited the most from attention is the Transformer [118] that uses the attention mechanism to improve both performance and efficiency. It was introduced in 2017 for the task of machine translation but has, since, been applied to various NLP tasks such as language modelling [2], language generation [94] and question answering [126]. The adoption of the Transformer model as a building block for many state-of-the-art models has established it as an architecture, rather than a model. In this section we present the core parts of the Transformer, since it is also employed in this work.

The Transformer, like RNNs, is designed to handle sequential data. In RNNs the data must be processed sequentially, in order to compute and feed back the hidden state. On the contrary, in the Transformer the sequential data can be processed in parallel, optimizing the

[20] In terms of complexity, this attention mechanism is similar to multiplicative attention, although Query-Key-Value attention can be implemented efficiently using optimized matrix multiplication code. Both variants perform similar for small dimensionality of the vectors, but additive attention performs better for larger dimensions. This is the intuition for scaling the dot-product.

training process and effectively increasing the amount of data that can be used for training. In order to compensate for the loss of order in sequential data, due to parallelization in the Transformer architecture, *positional encodings* are employed. These are, essentially, indexing vectors that are added to the input vectors to determine the position of each input in the sequence. The positional encodings can be fixed or learnable, in which case they are also called *positional embeddings*.

The Transformer has a sequence to sequence structure consisting of an encoder and a decoder. Both the encoder and the decoder contain multiple layers of the same sub-network, as illustrated in Figure 2.13. We note that the sub-networks for the encoder and the decoder differ and that the sub-network layers do not share parameters. The Transformer has been used as a building block for various models, either as a whole or partially, using only the encoder or the decoder.

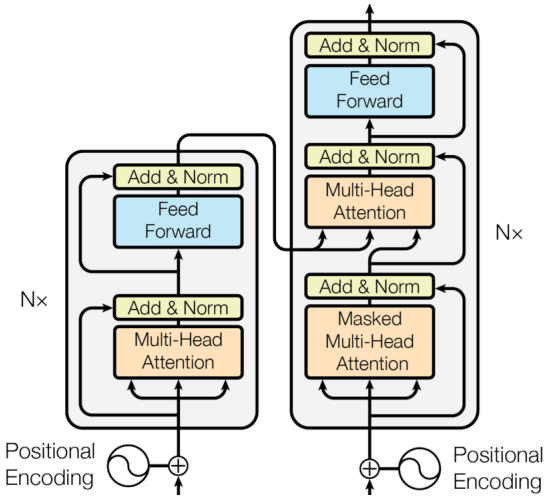


Figure 2.13: The Transformer architecture. Figure from [118].

Encoder: The encoder maps an input sequence $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$ to a continuous representation $\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_n}$. The core sub-network of the encoder consists of two components: a multi-head self-attention mechanism and a position-wise fully connected feed forward network. The multi-head attention mechanism refers to the Query-Key-Value attention with the extension of multiple attention heads, described in the previous section. Each point in the sequence is passed through the multi-head attention mechanism to obtain a representation that is related to both the current input and the rest of the sequence. Representations from multiple heads are then merged into a single representation[21]. The feed-forward network is applied to each point of the input sequence, independently. Each of the two components of the core sub-network (multi-head attention and feed-forward) is followed by a residual connection (described in 2.5.1) and layer normalization (described in 2.4.5). This subnetwork also called *Transformer layer* or *Transformer block* is illustrated in Figure 2.14. The output of each encoder layer is then passed as the input to the next encoder layer.

Decoder: The decoder takes as input encoded representations $\mathbf{z_1}, \mathbf{z_2}, \ldots, \mathbf{z_n}$ and generates an output sequence $\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_n}$, sequentially. The decoder has three components: a multi-head attention over the output of the encoder, a point-wise feed forward neural network and a masked multi-head attention mechanism. The multi-head attention of the

[21]The different context vectors $\mathbf{z_1}, \mathbf{z_2}, \mathbf{z_N}$ from the N attention heads are concatenated and reduced to a single vector via another projection $\mathbf{z} = \mathbf{W^O}[\mathbf{z_1}; \mathbf{z_2}; \ldots; \mathbf{z_N}]$.

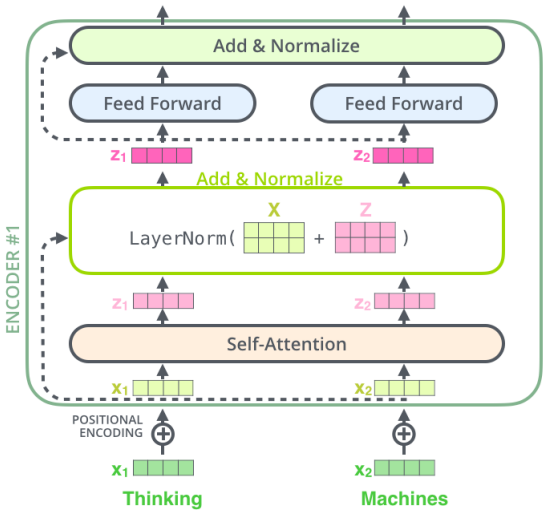


Figure 2.14: A single layer of the Transformer's encoder. Figure from `The Illustrated Transformer - Jay Alammar`'s blog. The vectors $\mathbf{x_1}$ and $\mathbf{x_2}$ compose the input sequence and the vectors $\mathbf{z_1}$ and $\mathbf{z_2}$ are the corresponding context vectors after the multi-head attention.

decoder is identical to the encoder attention mechanism, except that it is not self-attention, since the query vectors now come from the decoder and the key and value vectors from the encoder. The masked multi-head attention is similar to the encoder self-attention mechanism, with the addition of masks over the decoder representations[22] that are further in the sequence than the current position. The feed forward neural network, the residual connections and the layer normalization, are the same as in the encoder. The output of each layer of the decoder is, also, fed to the next layer and the final decoder layer is followed by a linear layer and a softmax in order to produce the probabilities over the vocabulary for the translation task.

2.6 Transfer Learning

In the supervised learning paradigm of section 2.2 we assumed that labeled data from a domain $\mathcal{D}$ are available in order to train a model for a task $\mathcal{T}$. However, this learning paradigm breaks down in the case that we do not have enough data to train a reliable model, especially given the fact that most deep learning models require large amounts of training data. A reasonable suggestion would be, to use a model that was trained to solve a different task, for the task we actually want to solve. In our analysis of empirical risk minimization in Section 2.3 we made the assumption that the test data comes from the same distribution as the training data, which would ensure that our model will generalize on unseen data. This, however, does not guarantee that our model will generalize on unseen data from a different distribution, as well. In order to overcome this challenge we employ *transfer learning*.

Before formally defining transfer learning we will introduce some concepts that will aid our analysis. A domain $\mathcal{D}$ consists of a feature space $\mathcal{X}$ and a marginal probability distribution $p(X)$ over the feature space, where $X = \mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n} \in \mathcal{X}$. Given a domain

[22] These representations are masked (set to $-Inf$) before the softmax of the attention mechanism.

$\mathcal{D} = \{\mathcal{X}, p(X)\}$, a task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a conditional probability distribution $p(Y|X)$[23] that is typically learned from the training data $\mathbf{x_i}, \mathbf{y_i}$, with $\mathbf{x_i} \in X$ and $\mathbf{y_i} \in \mathcal{Y}$, $\mathcal{T} = \{\mathcal{Y}, p(Y|X)\}$. Given a source domain $\mathcal{D_S}$ and corresponding task $\mathcal{T_S}$, a target domain $\mathcal{D_T}$ and corresponding task $\mathcal{T_T}$, transfer learning aims to help improve the learning of the target conditional probability $p(Y_T|X_T)$ in $\mathcal{D_T}$ using the knowledge in $\mathcal{D_S}$ and $\mathcal{T_S}$, where $\mathcal{D_S} \neq \mathcal{D_T}$, or $\mathcal{T_S} \neq \mathcal{T_T}$ [84].

The relation between source and target tasks, defines the different transfer learning settings. We will describe the two transfer learning scenarios that are employed in this work. We base our analysis in the transfer learning taxonomy introduced by Pan and Yang [84], illustrated in Figure 2.15.

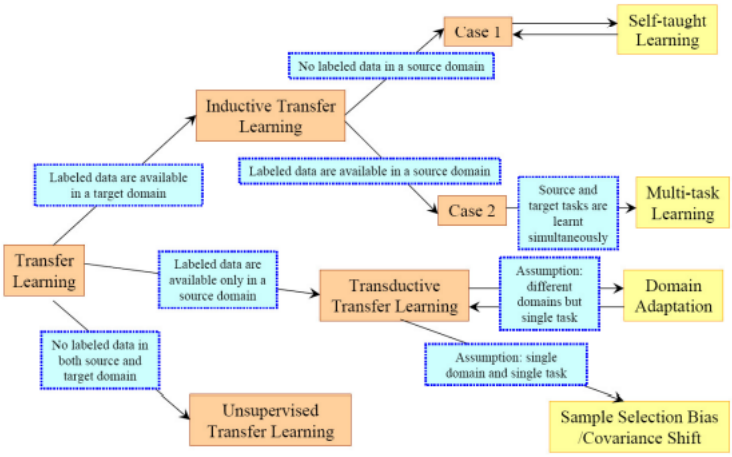


Figure 2.15: An overview of different settings of transfer learning. Figure from Pan and Yang [84].

The first transfer learning scenario we address in this work is *multi-task learning*. Multi-task learning is the setting where a model learns different tasks simultaneously and is a special case of *inductive transfer learning*. In the inductive transfer learning setting, the target task is different from the source task, $\mathcal{T_S} \neq \mathcal{T_T}$, no matter if the source and target domains are the same or not. This setting requires labeled data from the target domain. In the case that labeled data from the source domain are also available, namely the multi-task setting, the labels spaces are different $\mathcal{Y_S} \neq \mathcal{Y_T}$, which also implies that $p(Y_S|X_S) \neq p(Y_T|X_T)$.

In addition, this work is motivated by *domain adaptation* which is a category of *transductive transfer learning*. In the transductive transfer learning setting, the source and target task is the same, while the source and target domains are different. In the case of domain adaptation, labeled data exist only for the source domain. In this case the feature spaces between source and target domain are the same, $\mathcal{X_S} = \mathcal{X_T}$, but the marginal probability distributions of the input data are different $p(X_S) \neq p(X_T)$, which is also called *domain shift*.

2.6.1 Multi-task Learning

The definition of machine learning of section 2.1 implied that a model would learn from experience in order to perform a single task. However, the learning data could contain in-

[23] This is the case for the supervised learning paradigm.

formation related to other tasks, as well. This information, apart from its use in solving the other tasks, could improve the generalization of our model on the original task. This is an additional motivation for multi-task learning (MTL), specifically "*MTL improves generalization by leveraging the domain-specific information contained in the training signals of related tasks*" [14].

In practice MTL is the machine learning setting that involves more than one loss function, this is why it is also called *joint learning*. The other loss terms, also called *auxiliary losses* or *auxiliary tasks* may be of variable importance, relative to the main (source) task. This can be determined by weighting each loss based on its importance in the final loss function, which is also called *joint loss* function.

The two most widely used ways to perform multi-task learning in DNNs is via *hard parameter sharing* or *soft parameter sharing* of hidden layers. In soft parameter sharing, each task has its own model with its own parameters. The distance between the parameters of the model is then regularized in order to encourage the parameters to be similar [97]. In hard parameter sharing, many hidden layers are shared between all tasks, while there are some layers that are specific to each task, called *task-specific layers*. Hard parameter sharing is the most common practice in order to apply MTL in DNNs, and is also employed in this work.

Multi-task learning does not only aid in solving multiple tasks simultaneously but can also improve the performance of each task individually. One reason for the improved performance using MTL, could be the fact that the auxiliary tasks help the model focus on the input features, that are considered relevant for other tasks as well, which is potential evidence of the general importance of these features. In this way, the model can construct representations using the most important, overall, features, which can be seen as a form of regularization. In addition, some features could be easy to learn for some task $\mathcal{T}_{\mathcal{S}}$ but difficult to learn for another task $\mathcal{T}_{\mathcal{T}}$. MTL can improve the learning of task $\mathcal{T}_{\mathcal{S}}$ by additionally learning to solve the task $\mathcal{T}_{\mathcal{T}}$.

2.6.2 Domain Adaptation

Domain adaptation is of great practical importance for the application of machine learning models on real-world scenarios. Domain adaptation tackles the problem of using a model that was trained to solve a task on a source domain, $\mathcal{D}_{\mathcal{S}}$ to solve the same task on a different domain $\mathcal{D}_{\mathcal{T}}$, in the absence of labeled data for the latter.

Earlier works on domain adaptation used to change the representations of the data by identifying features that are common in both domains [12], [41] and/or projecting the source and target domain data in a shared space [39], [45], [111]. A deep learning approach to the previous idea is to minimize the distance of the source and target domains, using DNNs to acquire higher level representations of the source and domain data and then align these representations [70], [116].

A good representation for cross-domain transfer, according to the theoretical work on domain adaptation by Ben-David, Blitzer, Crammer, *et al.* [7], is one for which an algorithm cannot learn to identify the domain of origin of the input data. They therefore argue that a good representation would minimize the error on the particular task in the source domain and additionally minimize a distance metric between the source and target domains. This theoretical approach, inspired Ganin, Ustinova, Ajakan, *et al.* [36], who introduced adversarial training to domain adaptation, which was the clear motivation for this work.

Chapter 3

Natural Language Processing

3.1 Introduction

Natural language processing (NLP) can be defined as the field of AI that facilitates the interaction between humans and computers via natural language. This interaction requires that the computers are able to manipulate human language. To this end, NLP provides a set of models and techniques in order to extract meaningful information from natural language input and/or produce output in the form of natural language. In this section, we will analyze the most common approaches to NLP using the machine learning tools presented in Chapter 2. The concept of NLP can be further clarified via the tasks it tries to solve and its numerous applications.

3.2 Applications

Natural language processing combines the theory of linguistics and the mechanisms provided by ML in order to model both spoken and written language. In the context of this thesis, we are only concerned with the written form of human language, namely text. Applications of NLP to text involve extraction of specific attributes [15], [53], [83], translation to other languages [37], [118], [125], summarization [81], [87] and even complex question answering [105], [130] and dialogue systems [17], [106]. The various applications also define the different tasks that NLP tries to solve.

Natural language processing tasks can be divided into two broad categories: *natural language understanding* (NLU) tasks, where the goal is to extract meaning from natural language input; and *natural language generation* (NLG) tasks, that aim at producing natural language output, using computers. The first category can be viewed as reading or interpreting language while the second is focused on generating language. In this work, we apply the proposed fine-tuning approach on four NLU tasks: *sentiment analysis*, *linguistic acceptability*, *paraphrase detection* and *textual entailment*.

Sentiment Analysis: Sentiment analysis, or opinion mining, is the process of analyzing people's sentiments, emotions, evaluations and opinions towards entities such as products, organizations, individuals, events and topics. In the context of NLP, sentiment analysis is focused on extracting the sentiment of a text. The sentiment can be expressed as the general sentiment or opinion of a text, such as a document or sentence. In a more fine-grained analysis, both the sentiment and the particular target, towards which it is expressed, are extracted. Sentiment analysis is framed as a classification problem, with the number of classes depending on the particular dataset. The simplest case is a binary classification problem where the sentiment is either positive or negative, but it can be

extended with more classes that represent different emotions. Sentiment analysis has numerous real-world applications such as customer feedback, marketing surveys, advertising and public opinion monitoring that can be used for political campaigns.

Sentiment analysis can be regarded as a form of *text classification*, the NLP task that assigns a piece of text to a specific category, based on its content. The number and topic of the different categories can vary, based on the particular dataset. In sentiment analysis the categories are the different sentiments that can be expressed in text. Other text classification tasks include topic labelling, language detection and intent detection.

Linguistic Acceptability: Linguistic acceptability is the task that determines whether a sentence is grammatically acceptable. It is typically formed as binary a classification problem where each sentence is classified as grammatical or ungrammatical.

Paraphrase Detection: The task of paraphrase detection is to identify whether two sentences are semantically equivalent. In this case, the input sequence is not a single sentence but two different sentences. It is framed as a binary classification problem where the classes represent whether or not the first sentence is a paraphrase of the last.

Textual Entailment: Textual entailment, also known as *natural language inference*, is another task that is applied to a pair of sentences. The goal of this task is to determine whether the first sentence, called premise, entails, contradicts or is neutral to the second sentence, called hypothesis. We say that the premise entails the hypothesis if, typically, a human reading the premise would infer that the hypothesis is most likely true. Textual entailment is a classification problem where the various classes correspond to the possible cases. The "neutral" and "contradiction" class can be merged into a single, "not-entailment" class. In this case textual entailment is framed as binary classification (entailment/not-entailment) problem.

3.3 Language Representation

In order to solve the tasks that were introduced in the previous section, natural language must be represented in a form that is both informative and can be handled by computers. Language representation has been, historically, a challenging problem in NLP. A good numerical representation for language should not only uniquely identify each word, but should, ideally, convey meaning regarding its semantic and syntactic attributes, as well as the context in which it is produced.

In some fields of AI there exists a standard representation for the input data. In computer vision, for example, such a representation is the intensity of each pixel in an image. On the contrary, in NLP, a standard, unambiguous representation of words and sentences is not available, while even our knowledge of how language is produced and what rules apply to language generation is relatively limited.

An initial approach to language representation for ML models was to create a unique numerical vector for each word. These vectors would then be aggregated, in order to represent longer spans of language, such as phrases, sentences, paragraphs and documents. In the remaining section we will present different approaches to word and document representations.

3.3.1 Vocabulary

The first step towards language representation is the construction of a *vocabulary* V, given a corpus of different phrases, sentences or documents.

Words: In the simplest case, the vocabulary consists of a fixed number of words, $|V|$, that defines the number of possible words that can be represented by it. Words that do not belong in the vocabulary, namely *out-of-vocabulary* words (OOV), are therefore considered unknown. A vocabulary in this case, is usually formed by selecting the $|V|$ most frequent words in a corpus.

N-grams: Another approach to vocabulary construction is the use *n-grams* instead of words. An n-gram is a sequence of n consecutive words. In this case the vocabulary could consist of $|V|$ most frequent n-grams in the corpus. When $n = 1$, which is also called a *unigram*, this approach is equivalent to selecting words[1]. A benefit of n-grams is the fact that they capture some of the syntax and semantics of the words, since they encode a larger context. However, they do not solve the problem of large dependencies between words, which are frequent in text. In addition, the use of n-grams requires a larger vocabulary since there are more n-grams than words, due to the number of possible combinations of words.

Subwords: A completely different approach is to use *subwords* instead of words. A subword is a sequence of n consecutive characters, similar to a character n-gram. This representation technique allows the representation of more words using the same vocabulary size, reducing OOV words and effectively increasing the vocabulary. In the case, $n = 1$ the subwords are essentially characters. A subword vocabulary can be constructed using a fixed number of characters for all subwords [13], or with a varying subword size, based on more complicated segmentation algorithms [59], [101], [104].

3.3.2 Distributional Hypothesis

Various forms of language representations are based on the *distributional hypothesis*, which relates the distribution of words in a context with their corresponding meanings. According to this hypothesis, words in similar context would have similar meaning, or according to John Rupert Firth: "*you shall know a word by the company it keeps*". The direct implication of this hypothesis is that two words that are considered to be semantically similar are expected to occur in similar contexts, and vice-versa. Context is usually defined as a window of c words that surround a word, preceding and following, in a sentence.

3.3.3 Count-based Methods

A simple way to represent text in numerical form is *one-hot encoding*. Specifically, given a vocabulary of $|V|$ words, each word is represented as a vector of $|V|$ dimensions, with each dimension being associated with a different word. As a result, each word is represented as a vector containing 1 in the dimension related to the corresponding word and 0 in the other dimensions. However, this approach has many deficiencies, since one-hot vectors cannot encode semantic similarities between words, and a large vocabulary would lead to large word vectors which require larger models to handle them. Overall, one-hot encoding is inefficient for language representation, due to its sparsity, since it is composed of almost only zeros.

Bag-of-Words: An extension of one-hot encoding for sentences and documents is the *bag-of-words* (BoW) model. In this representation each document or sentence is represented by a vector, $\mathbf{d}$, of length $|V|$, where each dimension, d_i is associated with the presence of the word i in the document. The name bag-of-words comes from the fact that in this representation the order of the words does not matter.

[1] In the case $n = 2$ the n-gram is also called a *bigram* and in the case $n = 3$, a *trigram*.

In its simplest form, d_i could be a boolean variable indicating the presence (1) or the absence (0) of word i in the document. Alternatively, d_i could indicate the number of times the word i appears, or the corresponding frequency. This approach, however, suffers from the high frequency of non-informative words in documents, such as stopwords.

A more sophisticated approach is to rescale the frequency of words by how often they appear in all documents, so that the scores for frequent words that are also frequent across all documents are penalized. This is achieved by multiplying the *term frequency* of the word in the document with the *inverse document frequency* (TF-IDf), the inverse of the frequency of the word across all documents.

Co-occurrence Vector: A count-based approach to language representation that, additionally, leverages the distributional hypothesis can be achieved with the use of the co-occurrence of words. The implementation of this approach is based on the construction of a co-occurrence matrix, where the components of each vector can be interpreted as weights denoting the strength of the relationship between the target and the respective context word. A simple way to construct the co-occurrence matrix is by counting how often two words co-occur in a context of length c. In order to address problems that are related to the occurrence of frequent, uninformative words other scoring functions are employed, such as TF-IDF or *positive pointwise mutual information* (PPMI). Co-occurrence vectors are still sparse and, therefore, dimensionality reduction techniques are often applied to the co-occurrence matrix, such as SVD or PCA.

3.3.4 Prediction-based Methods

An alternative approach to representing language is to employ neural networks in order to acquire better word representations. Bengio, Ducharme, Vincent, *et al.* [9] proposed the use of neural networks in order to learn distributed representations of words. The neural network, in this case called *neural language model* would learn a distributed representation for each word. These dense representations of words in a low-dimensional space produced by neural networks are also called neural *word embeddings*.

The word embeddings are created using a neural network layer that takes as input words from the vocabulary and *embeds* them into a lower dimensional space. The embeddings are, in practice, the weights of this layer, also called *embedding layer* or *embedding matrix*, which is part of a larger network. The embedding matrix C maps any element i from the vocabulary V to a vector $C(i)$ of dimension m and has therefore $|V| \times m$ parameters.

Word2vec: One of the most popular word embedding models is the word2vec [76] model. Mikolov, Chen, Corrado, *et al.* [76] proposed two architectures for learning word embeddings that were computationally efficient and could thus leverage a large training corpus. The two architectures, illustrated in Figure 3.1, are approximations of language models on a fixed context.

The *Continuous Bag of Words* (CBOW) model architecture tries to predict the current target word (the center word) based on the source context words (surrounding words). The Skip-gram model architecture tries to achieve the inverse goal of the CBOW model. It tries to predict the source context words (surrounding words) given a target word (the center word). In both architectures, the embeddings are produced as the weights of the embedding matrix in the first layer.

Due to their efficient implementation strategies [78] and large training corpus, prediction-based vectors, such as word2vec, consistently outperform count-based vectors in almost all tasks [6]. Another interesting attribute of word embeddings is that in the low-dimensional space similar words tend to cluster together and also vector arithmetics can be employed

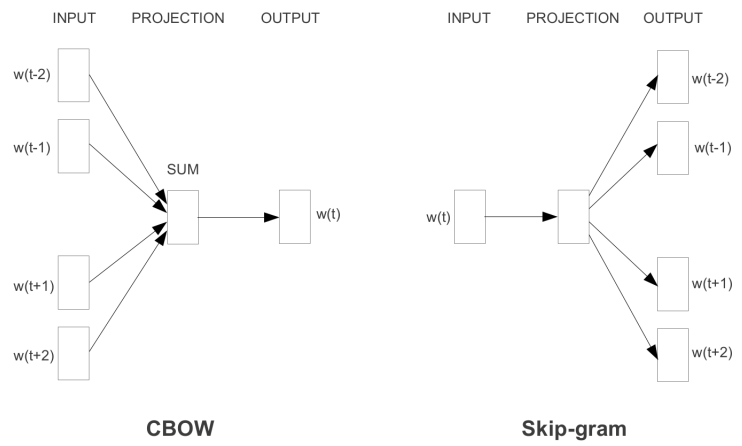


Figure 3.1: The two word2vec model architectures. Figure from [76].

in order to handle word relations (e.g. "king" is to "man" what "queen" is to "woman"), as seen in Figure 3.2.

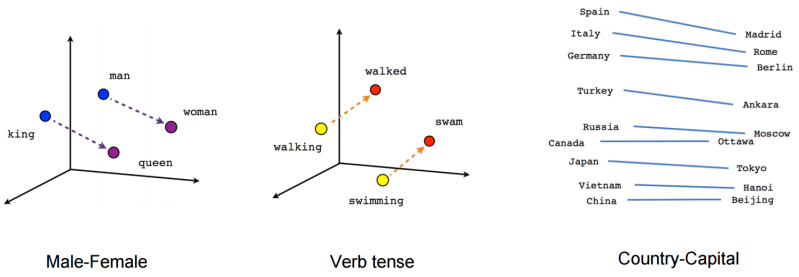


Figure 3.2: Relations between words in word2vec embeddings. Figure from Tensorflow word2vec tutorial.

3.4 Language Modeling

In the previous section we briefly introduced the concept of language models (LMs), that were used for the creation of word embeddings. *Language modelling* is the task of learning the joint probability function of sequences of words in a language, with a vocabulary V. Using the chain rule of probability, the probability estimation of a sequence of T words, $w_1, w_2, \ldots, w_T$ can be decomposed into the task of estimating the probability distribution over words in V for the next word w_i in the sequence, given the previous words:

$$P(w_1, w_2, \ldots, w_T) = \prod_{i=1}^{T} P(w_i | w_1, w_2, \ldots, w_{i-1}) \tag{3.1}$$

A fundamental problem in estimating the joint probability of a sequence words in equation 3.1, is that the number of possible configurations of words increases exponentially as the number of words increases, a problem known as the *curse of dimensionality*. Statistical language models used n-grams and *markov chains* of order n[2] in order to estimate

[2] A Markov chain of order n is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous n events.

the probabilities of equation 3.1. The probabilities of *n-gram language models* were computed based on the training corpus. As a result, these models suffered from the curse of dimensionality and were unable to generalize.

3.4.1 Perplexity

The evaluation of language models is done through *perplexity*, a measurement of how well a probability distribution or probability model predicts a sample. Formally, perplexity is defined as the exponential of cross entropy:

$$PPL(p,q) = 2^{H(p,q)} \ , \tag{3.2}$$

where the first distribution, p, refers to the empirical conditional distribution $\hat{p}_{data}(\mathbf{y}|\mathbf{x})$ and the second distribution, q, is the estimation of the true probability by our model, $p_{model}(\mathbf{y}|\mathbf{x};\boldsymbol{\theta})$ (see section 2.3.1). Since entropy is the average number of bits required to encode the information contained in a random variable, perplexity can be defined as the total amount of information contained in the random variable. Lower perplexity values indicate a better language model. Perplexity is equivalent to cross-entropy as an evaluation measure and its use in language modelling is mostly due to historical reasons.

3.4.2 Neural Language Models

Bengio, Ducharme, Vincent, *et al.* [9] proposed a neural language model that would simultaneously learn the word embedding for each word and a probability function of a sequence of words, expressed in terms of these embeddings. This model would generalize by constructing similar embeddings for similar words and leveraging the smoothness prior (see section 2.3) for the estimated probability function. As a result, a sequence of words that was not seen during training would get high probability if it is made of words that are similar to words forming a sequence present in the training corpus. This neural language model is illustrated in Figure 3.3.

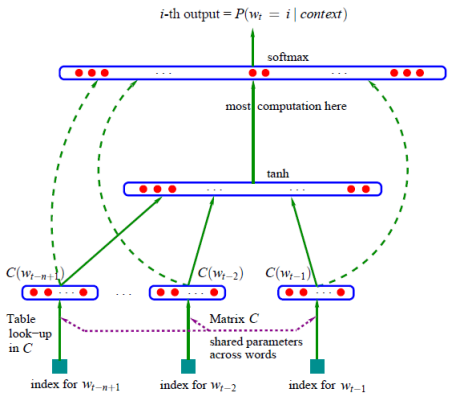


Figure 3.3: The standard neural language model architecture. Figure from [9]

The basic building-blocks of the neural language model of Figure 3.3 are still present in current state-of-the-art LMs. This model takes as input the n previous words and maps them to their corresponding m-dimensional embeddings $C(w)$, $C \in \mathcal{R}^{|V|\times m}$. The embeddings are then fed into one or more hidden layers that produce an intermediate representation of the input (e.g. a fully-connected layer that applies a non-linearity to the

concatenation of the word embeddings). Finally, the intermediate representation is fed to a *softmax layer* that produces a probability distribution over words in V.

Softmax layers are used for classification tasks, such as language modelling, in order to produce a probability distribution over all classes. A softmax layer is essentially a fully connected layer with a softmax activation function. The fully connected layer contains a weight matrix, $W \in \mathcal{R}^{d \times C}$, where d is the number of dimensions of the output of the penultimate network layer (the last hidden layer before the softmax layer) and C is the number of classes. We observe that, in an LM, the weight matrix has the same dimensions as the embedding matrix. These two matrices can be tied (the embedding matrix is the transpose of the softmax layer matrix) in order to reduce the total parameters of the model, which can also lead to improved performance [74], [92].

In recent extensions of the standard neural language model, FFNNs have been replaced by RNNs [77], LSTMs [74], LSTMs with attention mechanisms [75] and even Transformers [2], [27]. Current LMs operate at the word level [27], [74], at the subword level [28], [126] or even the character level [2]. In the case of subwords, the sequence of words is transformed into a sequence of *tokens*.

3.4.3 Language Modelling Objectives

Standard LM objective: Neural language models are trained to maximize the average log-likelihood of the training corpus, using the chain rule of probability. The sequence of equation 3.1 is often further decomposed into subsequences, leveraging markov chains of order n, due to the inherent limitations of models to handle very large sequences:

$$L(\boldsymbol{\theta})^{3} = \frac{1}{T} \log p_{model}(w_1, w_2, \ldots, w_T; \boldsymbol{\theta}) \approx \frac{1}{T} \sum_{i=1}^{T} \log p(w_t | w_{t-1}, \ldots, w_{t-n+1}) \tag{3.3}$$

As we discussed in section 2.3.1, this is equivalent to minimizing the cross entropy between the empirical distribution and the estimation of the true probability by our model, $H(\hat{p}_{data}, p_{model}; \boldsymbol{\theta})$. Due to the formulation of equation 3.3, the LMs that are trained with this objective are *autoregressive*, since each output variable (word w_i) depends linearly on its own previous values ($i-1$ words) and on a stochastic term (neural LM). We also observe that standard language modelling is framed as a self-supervised classification task (see section 2.2.3), since the labels for each training sample (the next word w_i in the sequence), are generated from the samples of the same dataset (sequence of $i-1$ words). Standard language modelling is also named *conditional* or *causal language modelling* (CLM).

Variants of the LM objective: In section 3.3.4 we mentioned that the two word2vec architectures are an approximation of language models. For example, the CBOW model is trained to maximize a variant of the standard language modelling objective:

$$L(\boldsymbol{\theta}) = \frac{1}{T} \sum_{i=1}^{T} \log p(w_t | w_{t-1}, \ldots, w_{t-n}, w_{t+1}, \ldots, w_{t+n}) \tag{3.4}$$

The standard language modelling objective can be also modified to leverage both past and present information to make a prediction about a token using bidirectional language

[3] We use $L(\boldsymbol{\theta})$ to refer to the per instance loss of LMs, which is, in this case, a per sequence of words $\boldsymbol{w} = w_1, w_2, \ldots, w_T$ loss : $L(f(\boldsymbol{w}; \boldsymbol{\theta}), \boldsymbol{w})$, where $f(\boldsymbol{w}; \boldsymbol{\theta}) = p_{model}(\mathbf{w}; \boldsymbol{\theta})$.

models (LSTMs) that process input text from left-to-right and from right-to-left [89]:

$$L(\boldsymbol{\theta}) = \frac{1}{T}\sum_{i=1}^{n}\Big(\log p(w_i|w_1,\ldots,w_{i-1};\boldsymbol{\theta}_e,\overrightarrow{\boldsymbol{\theta}}_{\text{LSTM}},\boldsymbol{\theta}_s)+ \tag{3.5}$$

$$\log p(w_i|w_{i+1},\ldots,w_n;\boldsymbol{\theta}_e,\overleftarrow{\boldsymbol{\theta}}_{\text{LSTM}},\boldsymbol{\theta}_s)\Big)\ , \tag{3.6}$$

where $\boldsymbol{\theta}_e$ is the embedding matrix, $\boldsymbol{\theta}_s$ is the softmax layer and $\overrightarrow{\boldsymbol{\theta}}_{\text{LSTM}}$, $\overleftarrow{\boldsymbol{\theta}}_{\text{LSTM}}$ are the LSTM parameters in the forward and the backward direction, respectively. We notice that in this implementation [89], the softmax and embedding matrix parameters are tied, while the LSTM parameters are separate for each direction. However, this is not always the case, since the forward and backwards LM can be trained independently [51].

MLM objective: Standard LMs, can be trained with either a left-to-right or a right-to-left LM objective since the use of both directions would allow each word to indirectly "see itself" in a multi-layered context. In order to mitigate this effect Devlin, Chang, Lee, *et al.* [28] proposed a variant of the standard LM objective, called *masked language modelling* (MLM), that enables the use of bidirectional context in LM training. For the MLM objective, a percentage of the input tokens are randomly masked and then only the masked tokens are predicted, as illustrated in Figure 3.4. The MLM objective can be regarded as an extension of the CBOW objective, using larger context and multiple predictions per sequence.

Figure 3.4: Forward, backward and masked language modelling. Figure from Paper Dissected: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" Explained - Machine Learning Explained blog.

This LM objective, which is similar to a *cloze* test [115], can be viewed as a denoising task, where we inject noise (masks) to the input signal (sequence of words/tokens) and then we must predict part of the original input. However, if the majority of the tokens are masked, then there is not enough context to accurately predict the original tokens. On the other hand, if the percentage of masked tokens is low, then the training procedure is more expensive since it requires more iterations in order to make sufficient predictions. In the original work [28] the percentage of masked tokens per sequence was 15%. The MLM objective has been reported to outperform the standard LM (CLM) objective [23], [28] due to its ability to leverage bidirectional context and has been adopted by many state-of-the-art, Transformer-based LMs [28], [62], [69]. Finally, the combination of the MLM and CLM objectives has been reported to improve performance [31], [96].

Although the MLM objective has resulted in state-of-the-art performance it has certain limitations. First, the masking of the input sequence occurs only during the minimization of the MLM objective, creating a discrepancy between the input signal of MLM (masked and unmasked tokens) and every other objective (unmasked tokens only). In addition, each masked token is predicted independently of the other masked tokens (i.e. the masked tokens are considered independent given the unmasked tokens), thus the model is not able

to model the joint probability of equation 3.1, factorized by the probability chain rule. This leads to a weaker training signal since the LM does not model dependencies between the masked tokens. In order to address these limitations Yang, Dai, Yang, *et al.* [126] proposed *permutation language modelling* (PLM), that maximizes the expected log-likelihood of a sequence (equation 3.3) w.r.t. all possible permutations of the factorization order. The PLM objetive leverages bidirectional context without the independence assumption and the input noising of the MLM objective. Similar to BERT, the authors of XLNet predict only the last tokens of any factorization order to leverage enough context and reduce optimization difficulties.

3.5 Transfer Learning in NLP

In section 2.6 we introduced the concept of transfer learning, where the knowledge acquired from solving a source task $\mathcal{T}_S$ from domain $\mathcal{D}_S$ is used to solve a target task $\mathcal{T}_T$ from domain $\mathcal{D}_T$, where $\mathcal{T}_S \neq \mathcal{T}_T$ or $\mathcal{D}_S \neq \mathcal{D}_T$. Transfer learning has had a huge impact in NLP, leading to numerous models that have achieved state-of-the-art results in downstream tasks. The transfer learning scenario that prompted these state-of-the-art results in NLP is *sequential tranfer learning* [97]. In sequential transfer learning, the source and target tasks are different and training is performed in sequence. Models are not optimized jointly as in multi-task learning but each task is learned separately [97]. Sequential transfer learning consists of two steps: a *pretraining* step, where a model is trained on the source task, and a *fine-tuning* step, where the model is adapted to perform the target task. The fine-tuning step typically consists of task-specific modifications to the original model and further training on the target task data. The benefit of this transfer learning scheme is that, usually, the source task contains much more data than the target task. Thus, although the pretraining step is computationally expensive, it needs to be performed only once. In NLP, the most common source task is language modelling which can leverage the abundance of unlabeled corpora, due to its self-supervised nature.

3.5.1 Pretrained Word Embeddings

A simple application of transfer learning and particularly sequential transfer learning in NLP is the use of language representations from pretrained models. This is the case for pretrained word embeddings such as word2vec [76] and GloVe [88], that are used to initialize the first layer of a task-specific model. The embeddings can then be trained along with the rest of the model or remain "frozen" during the fine-tuning process. This used to be the most common transfer learning scheme in NLP that resulted in improved performance, since the pretraining of word embeddings in a large corpus would capture some of the syntax and semantics of the words.

3.5.2 Contextualized Word Embeddings

Although pretrained word embeddings were widely used in NLP due to the ease of implementation and performance boost, this transfer learning scheme has inherent limitations. The embeddings for the words are the same, irrespective of the context in which they are used. This context-independent representation of words fails to capture language concepts that require higher-level reasoning such as long-term dependencies, polysemy and negation, notions that are related to *language understanding*.

In order to address these limitations and leverage language representations that are both semantically meaningful and contextual, extensions to the word embeddings transfer

learning scheme have been proposed. Instead of initializing the first layer of a model with pretrained word embeddings, the language representations are produced from a contextual language model, that was trained on a large corpus [72], [89]. The contextualized representations, named *contextualized word embeddings*, are higher level representations that are derived from the pretrained model. Given a sequence of words, contextual embeddings are first generated with the use of the pretrained model and are, then, used as features for the target task model.

3.5.3 Pretrained Language Models

The performance of task-specific models improved with the use of contextual word embeddings over the standard word embeddings, but was still constrained by the architecture that was employed for the particular target task. An extension of transferring embeddings, intermediate representations and weights from a pretrained LM to a new task-specific architecture, is to use the pretrained LM for the target task, as well [26]. The goal of this transfer learning scheme is to transfer the higher-level semantics captured during the LM pretraining.

ULMFit: To this end, Howard and Ruder [51] introduced a transfer learning method that involved pretraining an LM on a large, generic unlabeled corpus (e.g. Wikipedia) and then fine-tuning the same LM to solve various NLP tasks, one task at a time. The proposed method, named *Universal Language Model Fine-Tuning* (ULMFiT), achieved competitive performance on various NLP tasks, outperforming complex task-specific architectures. The results of a pretrained LM with minimal modifications for each task demonstrated the benefits of sequential transfer learning and large-scale LM pretraining.

OpenAI's GPT: Radford, Narasimhan, Salimans, *et al.* [94] also reported improved results on NLP tasks by pretraining an LM on diverse corpora and then fine-tuning it on each task. Furthermore, they used a Transformer decoder architecture for the proposed LM, named *GPT*. In order to achieve effective transfer without extensive changes to the LM architecture across tasks, they additionally used task-aware input transformations during fine-tuning.

These works demonstrated the effectiveness of LM pretraining as a good initialization point for task-specific models and paved the way for the use of pretrained LMs in most NLP tasks, which is now standard practice [28], [62], [69], [95], [96]. In addition, the use of the Transformer architecture has enabled the scaling of LMs, and, as a result, recent state-of-the-art LMs keep growing in depth and number of parameters, as illustrated in Figure 3.5. These pretrained LMs, do not focus on the language modelling task, but leverage the general-domain LM pretraining with a view to solve other NLP tasks, when fine-tuned with minimal effort. In practice, the success of these models is not measured by means of perplexity on standarized language modelling datasets (many state-of-the-art LMs do not even report such numbers). Instead, these LMs are mostly pretrained using a large amount of online text corpora (e.g. Wikipedia, fiction books, forums, etc.) and their success is determined by their results when fine-tuned on downstream tasks.

BERT: Inspired by the above findings, Devlin, Chang, Lee, *et al.* [28] proposed a language model that employed a Transformer encoder architecture and was pretrained on a very large corpus, named *BERT*, short for Bidirectional Encoder Representations from Transformers. BERT leverages the bidirectional nature of the Transformer encoder to effectively process input text, resulting in a significant performance boost. A possible reason for the success of BERT is its syntax-aware attention. Results from analysing BERT's attention heads suggest that BERT learns some aspects of the syntax as a byproduct of its

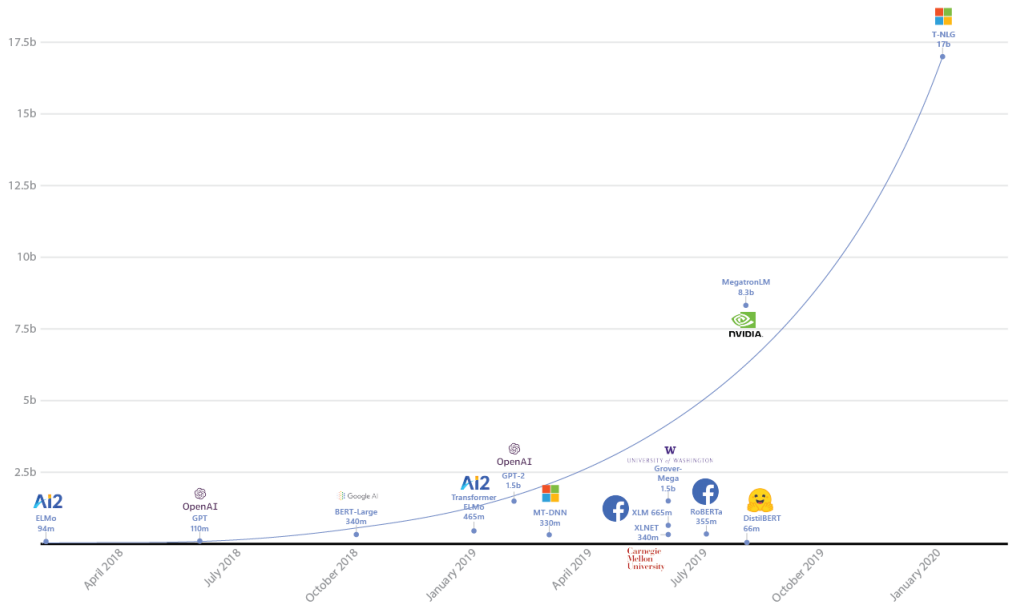


Figure 3.5: Parameter counts of several recently released pretrained LMs. Figure from Turing-NLG - Microsoft Research Blog blog.

self-supervised training [19]. BERT achieved state-of-the-art results in many NLP tasks, is, in addition, employed in this work and will be presented in this section.

To encourage the bidirectional processing of input sequences, BERT is pretrained using the MLM objective, presented in section 3.4.3. The token masking is achieved by randomly replacing tokens with the `[MASK]` token. However, this token is only encountered during the pretraining step, creating a train-test discrepancy. To counter this, the authors replaced the chosen tokens (to be predicted) with the `[MASK]` token only on 80% of the times. They additionally replaced the chosen tokens with a random token 10% of the times and the rest 10% they left the tokens unchanged.

The authors, additionally, introduced the *Next Sentence Prediction* (NSP) task during pretraining, to aid sentence-level understanding, which is essential for most NLP tasks. For the NSP task, the model is provided with two different segments that are identified using different segment embeddings, similar to positional embeddings of section 2.5.5. A binary classifier is, then, employed to predict whether the second sentence is the actual sentence that follows the first. Although, this task was originally reported to improve performance, its effect has been a matter of controversy in subsequent works [69], [126].

The input representation of BERT is illustrated in Figure 3.6. The input is the sum of the (learned) positional embeddings, the segment embeddings, mentioned above, and the token embeddings. The first token of the input sequence is always the `[CLS]` token, that is also used for the NSP task during pretraining. The segments are additionally separated using the `[SEP]` token. We also note that BERT uses subword tokens (e.g. playing is split to two different tokens: "play" and "##ing" [4]) that are created using a *WordPiece* model [102]. The wordpiece model is generated using a data-driven approach to maximize the language-model likelihood of the training data, given an evolving word definition. Given a training corpus and a number of desired tokens D, the optimization problem is to select D wordpieces such that the resulting corpus is minimal in the number of wordpieces when

[4]The double hash marks (##) are used to indicate that the subword is not the first token of the word.

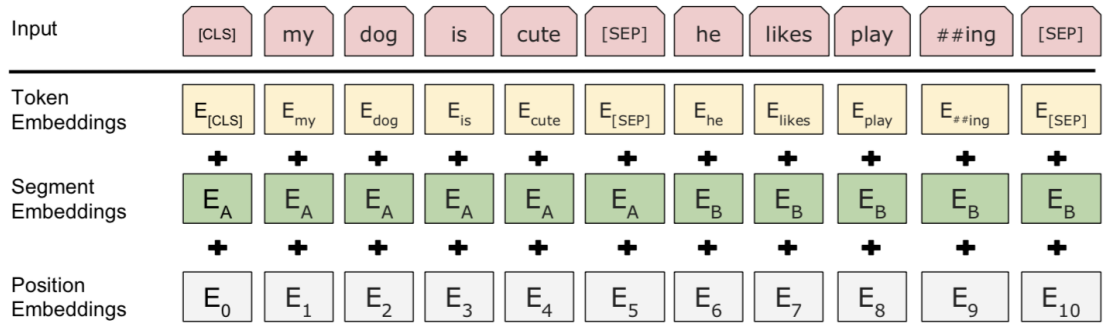


Figure 3.6: The input representation of BERT. Figure from [28].

segmented according to the chosen wordpiece model [125]. Thus, the tokens generated by the wordpiece model are a middle ground between characters and words, creating word level inputs for frequent symbol sequences and character level inputs for infrequent symbol sequences. BERT leverages a vocabulary of $30K$ wordpieces.

The use of BERT for a particular target task requires minimal changes to the initial architecture. Specifically, for text classification tasks, which are the focus of this work, a softmax layer is added on top of the final hidden state of the `[CLS]` token, as illustrated in Figure 3.7. The classifier is fine-tuned along with the pretrained model using the same learning rate.

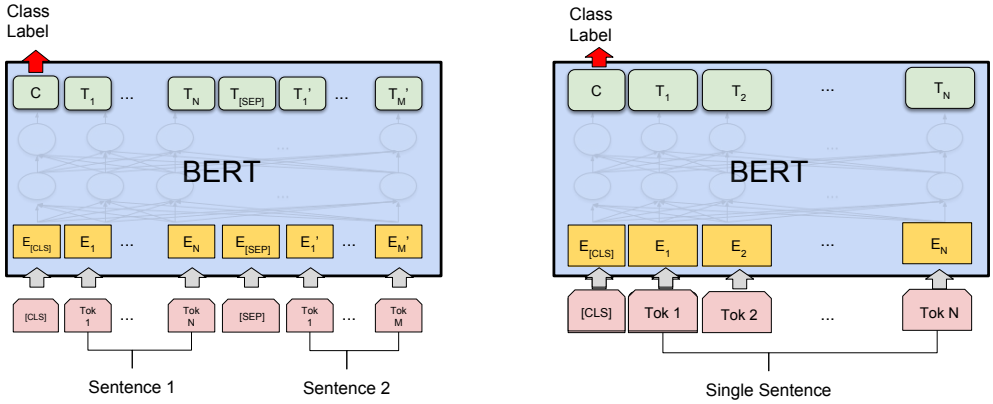


Figure 3.7: The modifications to BERT for sentence-pair (Left) and single-sentence (Right) classification tasks. Figure from [28].

Devlin, Chang, Lee, *et al.* [28] trained two models of different sizes in order to evaluate the results on the downstream tasks:

- BERT-BASE, which has $L = 12$ layers (Transformer blocks), $A = 12$ attention heads, feed-forward network output dimensionality $d_{ff} = 3072$ (see section 2.5.5), query, key and value vectors of dimension $d_k = d_v = 64$, $H = 768$ hidden size (all other sub-layers and embeddings), with a total of 110 million parameters and
- BERT-LARGE, which has $L = 24$ layers, $A = 16$ attention heads, $d_{ff} = 4096$, $d_k = d_v = 64$, $H = 1024$ hidden size, with a total of 340 million parameters.

XLNet: To address the limitations of the MLM objective, Yang, Dai, Yang, *et al.* [126] proposed *XLNet*, a new pretraining method that uses permutation language modelling

(PLM), paired with enhancements to the Transformer encoder architecture. Although XLNet employs a similar architecture to that of BERT, it is slightly modified to handle the PLM objective and leverage longer context.

The PLM objective combines the benefits of autoregressive models (trained with standard LM objective) and bidirectional context. As presented in section 3.4.3, the goal of the PLM objective is to maximize the expected log-likelihood of a sequence for all possible permutations of the factorization order. For a sequence of length T there are $T!$ different factorization orders. In practice, the authors propose to randomly sample a factorization order and decompose the likelihood of equation 3.3 according to this factorization order. An illustration of the two pretraining tasks of BERT and XLNet for a sample sentence can be found in Figure 3.8.

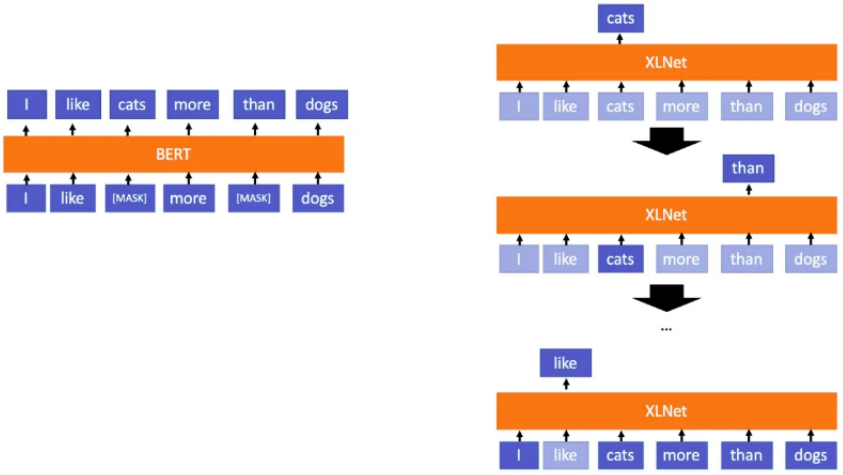


Figure 3.8: The difference between the pretraining tasks of BERT-MLM (Left) and XLNet-PLM (Right). Figure from Paper Dissected: "XLNet: Generalized Autoregressive Pretraining for Language Understanding" Explained - Machine Learning Explained blog.

For the permutation of the PLM objective the original sequence order is preserved and only the order of the predicted words is changed, as can be seen in Figure 3.8. As a result, the positional embeddings are retained and the permutation is realized through proper masking of the attention mechanism. In the original implementation of the MLM objective [28], the positional embeddings of the masked tokens were masked as well, resulting in position-agnostic token prediction. This limitation of the model to learn useful representations for the predicted tokens is more evident with the PLM objective where partially similar permutations (e.g. $cats \rightarrow than \rightarrow dogs$ and $cats \rightarrow than \rightarrow other$ in the example of Figure 3.8) would result in the same prediction.

To tackle this limitation, the authors also proposed a *Two-Stream Self-Attention* mechanism during pretraining that uses two sets of representations for all tokens. The original set of representations that includes the positional and token embeddings is called the *content stream*. The second set of representations incorporates positional information for the current token and contextual information for all the previous (in the selected factorization order) tokens. Consequently, each target token is both position-aware and has access to the context of the previous tokens. This is realised through proper masking of the attention mechanism, as well. We refer the reader to original work [126] for a more detailed presentation of the particular attention mechanism.

Aside from using permutation language modeling, XLNet improves upon BERT by

using the *Transformer-XL* [27] architecture. The Transformer-XL extends the standard Transformer architecture by introducing recurrence at the segment level to capture longer contexts. Specifically, the hidden representations computed for the previous segment are cached, in order to be reused as additional context when the model processes the next new segment. XLNet also employs the relative positional embeddings of Trasnformer-XL that represent the relative (instead to the absolute) distance between tokens to enable the reuse of tokens from other segments and generalize to arbitrary lengths. In addition, XLNet uses relative segment embeddings that generalize to more than two segments.

Similar to BERT, XLNet has a vocabulary of $32K$ subword tokens that is created using a wordpiece model. The input sequence of XLNet also contains a `[CLS]` token and therefore the input representation and fine-tuning procedure is the same as BERT. However, the authors of XLNet did not find the Next Sentence Prediction task to improve performance on downstream tasks and thus it was not included during the pretraining of the model. On par with BERT, two sizes of the XLNet model are available: XLNet-base and XLNet-large. With the use of the PLM objective, the Transformer-XL architecture and the corresponding modifications XLNet managed to outperform BERT in many NLP tasks and is additionally employed in this work.

3.5.4 Other pretraining tasks

Apart from language modelling, other pretraining task have also been proposed for sequential transfer learning in NLP, such as machine translation [72] and natural language inference [22]. However, language modelling can be still viewed as an ideal source task since it captures many facets of language that are relevant for downstream tasks such as sentiment [93] and syntax [132]. Furthermore, it provides training data in near unlimited quantities for most domains and languages, which makes it a very appealing pretraining task.

3.6 Evaluation Metrics

In order to evaluate and compare different models we employ performance measures that are specific to each task. These measures are usually different from the loss functions that are used during training, as described in section 2.3.1. In the academic setting, the performance metrics are reported on the test sets of standard benchmarks, to facilitate an even comparison between models and techniques. In this section, we present some evaluation metrics that are used for classification tasks, which are common in NLP and are also addressed in this work.

For a proper definition of the performance metrics, we consider useful to first introduce some concepts for the case of binary classification. In the binary classification setting, *True Positives* (TP) are the samples correctly labeled as belonging to the positive class, while, on the other hand, *False Positives* (FP) are the samples incorrectly labeled as belonging to the positive class (while belonging in the negative class). The same reasoning applies to *True Negatives* (TN) and *False Negatives*. A tool that is commonly used to visualize the performance of a classification system for the binary case is the *confusion matrix*, illustrated in Table 3.1:

The *accuracy* of a classifier can be, then, defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{3.7}$$

		Predicted Class: Class 0	Predicted Class: Class 1
Actual Class	**Class 0**	**TP**	FP
	Class 1	FN	**TN**

Table 3.1: Confusion matrix for binary classification. **Bold** indicates correctly labeled samples.

where the denominator is the total number of dataset samples. Intuitively, accuracy is the percentage of correct predictions over all predictions. However, accuracy can be a misleading metric for imbalanced datasets. For example, for a dataset with 95 positive and 5 negative samples, classifying all samples as belonging to the positive class, gives 0.95 accuracy score. We, therefore, use alternative metrics:

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN}, \quad F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3.8}$$

Intuitively, *precision* is the fraction of the samples that were correctly labeled, among all the samples that were classified as positive. *Recall* is the fraction of the actual positive samples that were correctly classified as belonging to the positive class. These metrics are, additionally illustrated in Figure 3.9. Finally, *F1* combines the two metrics, since it is the harmonic mean of precision and recall.

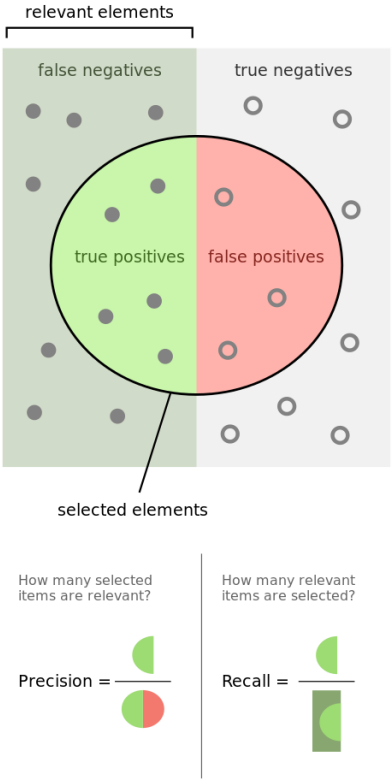


Figure 3.9: Illustration of precision and recall metrics. Image taken from Precision and recall - Wikipedia.

The accuracy metric can be generalized for the multi-class scenario by simply computing the average number of correct predictions (average over accuracy for each class). For

the precision and recall metrics, there are two ways of handling the multi-class setting: *micro-averaging* and *macro-averaging*. The micro-average of these metrics is the average performance over observations while macro-average is the average performance over classes. The macro-average is therefore more suitable for imbalanced datasets and is thus, employed in this work:

$$Precision_{\text{macro}} = \frac{1}{C}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i + FP_i} = \frac{\sum_{i=1}^{C} Precision_i}{C} \tag{3.9}$$

$$Recall_{\text{macro}} = \frac{1}{C}\sum_{i=1}^{|C|}\frac{TP_i}{TP_i + FN_i} = \frac{\sum_{i=1}^{C} Recall_i}{C}\ , \tag{3.10}$$

where C is the number of classes. The F1 measure for multi-class problems is computed using either micro-averaged or macro-averaged precision and recall. In this work, we employ macro-average for F1, as well. A large $F1_{macro}$ value indicates that a model performs well for each individual class:

$$F1_{\text{macro}} = 2 \cdot \frac{P_{\text{macro}} \cdot R_{\text{macro}}}{P_{\text{macro}} + R_{\text{macro}}}\ , \tag{3.11}$$

where P_{macro} and P_{macro} are abbreviations for $Precision_{\text{macro}}$ and $Recall_{\text{macro}}$, respectively.

Although F1 is a more balanced metric compared to accuracy, precision and recall it could still produce biased results since it does not take into account TN from the confusion matrix 3.1. A more balanced metric that takes into account all cells from the confusion matrix is *matthews correlation coefficient* (MCC):

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{3.12}$$

MCC is a correlation coefficient between the predicted and the actual classes of the samples. Thus, matthews corr. values range from -1 to $+1$, with $+1$ indicates perfect correlation, -1 perfect disagreement and 0 no correlation at all. In contrast to precision, recall and F1, MCC leverages all the values from the confusion matrix. In addition, a high value of MCC means that both classes are predicted well irrespective of class imbalance, unlike accuracy. Thus, MCC is often a preferable metric for binary classification problems.

Chapter 4

Domain Adversarial Fine-Tuning as an Effective Regularizer

In Natural Language Processing (NLP), pretrained language models (LMs) that are transferred to downstream tasks have been recently shown to achieve state-of-the-art results, as presented in Chapter 3. In this work, we extend the standard fine-tuning process of pretrained LMs by introducing a new regularization technique, AFTER; domain Adversarial Fine-Tuning as an Effective Regularizer. Specifically, we complement the task-specific loss used during fine-tuning with an adversarial objective. This additional loss term is related to an adversarial classifier, that aims to discriminate between *in-domain* and *out-of-domain* text representations. In-domain refers to the *labeled* dataset of the task at hand while out-of-domain refers to *unlabeled* data from a different domain. Intuitively, the adversarial classifier acts as a regularizer which prevents the model from overfitting to the task-specific domain. Empirical results on sentiment analysis, linguistic acceptability, textual entailment and paraphrase detection show that AFTER leads to improved performance compared to standard fine-tuning.

4.1 Motivation

Current research in NLP focuses on transferring knowledge from large language models (LMs), pretrained on large general-domain data, to a target task. The LM representations are transferred to the target task either as additional features of a task-specific model, as in ELMo [89], or typically, by fine-tuning, as in ULMFit [51], OpenAI GPT [94], BERT [28] and XLNet [126]. Standard fine-tuning involves initializing the target model with the pretrained LM and further training it with the target data.

Fine-tuning, however, can lead to catastrophic forgetting [43], if the pretrained language representations are adjusted to such an extent to the target task, that most generic knowledge, captured during pretraining, is in effect forgotten. In addition, pretrained LMs tend to overfit when fine-tuned to a target task dataset that is several orders of magnitude smaller than the pretraining data [26]. Thus, the fine-tuning process is generally performed only for a few optimization steps with careful regularization.

Adversarial training is a method to increase robustness and regularize deep neural networks [44], [80]. It has been used for domain adaptation [36] to train a model from scratch to produce representations that are invariant to different domains. Inspired by this approach, we propose a regularization technique for the fine-tuning process of a pretrained LM, that aims to optimize knowledge transfer to the target task and avoid overfitting.

Our method, domain Adversarial Fine-Tuning as an Effective Regularizer (AFTER)

extends standard fine-tuning by adding an adversarial objective to the task-specific loss. We leverage out-of-domain *unlabeled* data (i.e. from a different domain than the target task domain) and fine-tune the transferred LM so that an adversarial classifier cannot discriminate between text representations from in-domain and out-of-domain data. This loss aims to regularize the extent to which the model representations are allowed to adapt to the target task domain. Thus, AFTER is able to preserve the general-domain knowledge acquired during the pretraining of the LM, while adapting to the target task. We show that AFTER improves the performance of standard fine-tuning in four natural language understanding tasks from the GLUE benchmark [120], with two different pretrained LMs: BERT [28], and XLNET [126].

The remainder of this chapter is organized as follows: In the next section (sec. 4.2) we compare the proposed approach to related work. In section 4.3 we present AFTER, a method that aims to avoid catastrophing forgetting of general-domain knowledge, acting as a new kind of regularizer. In section 4.4 we present the datasets used and provide implementation details. In section 4.5 we provide empirical results of the proposed fine-tuning technique on four tasks and two pretrained LMs. Finally, in section 4.6, we conduct an ablation study of our approach to provide useful insights regarding the key factors of the proposed approach.

4.2 Related Work

In section 2.6.2 we introduced several approaches that have been proposed for the adaptation of a model trained on a source domain $\mathcal{D}_S$ to a different domain $\mathcal{D}_T$, where no labeled data is available [45], [111], [116]. We additionally presented the work of Ganin, Ustinova, Ajakan, *et al.* [36] that applies the theoretical work of Ben-David, Blitzer, Crammer, *et al.* [7] by proposing an adversarial method for domain adaptation.

Szegedy, Zaremba, Sutskever, *et al.* [114] introduced adversarial examples as inputs that are formed by applying intentional, small perturbations which cause an ML model to make false predictions. Adversarial examples can be incorporated in the training procedure, a process known as *adversarial training*, in order to provide additional regularization and increase the robustness of the model [44], [79]..

Ganin, Ustinova, Ajakan, *et al.* [36] were the first to introduce adversarial training for *domain adaptation*. They proposed a gradient reversal layer (`GRL`) to adversarially train a classifier that should not be able to discriminate between $\mathcal{D}_S$ and $\mathcal{D}_T$, in image classification and sentiment analysis tasks. The `GRL` acts as an identity transform during the forward pass, but during backpropagation, it *reverses* the incoming gradients, as illustrated in Figure 4.1. While Ganin, Ustinova, Ajakan, *et al.* [36] use a domain adversarial objective to train a model from scratch for effective domain adaptation, we introduce this objective as an auxiliary loss in order to regularize the fine-tuning process of a pretrained model.

Various adversarial losses have been used for domain adaptation in several NLP tasks, such as question answering [65], machine reading comprehension [121] and cross-lingual named entity recognition [55]. Adversarial approaches have been also used to learn latent representations that are agnostic to different attributes of the input text, such as language [60], [61] and style [127]. Contrary to previous *domain adaptation* work, we explore the addition of an adversarial loss term to serve as a *regularizer* for fine-tuning.

Other variants of LM fine-tuning include a supplementary supervised training stage in data-rich tasks [90] or multi-task learning with additional supervised tasks [68]. However, such methods require additional *labeled* data. A common way to leverage *unlabeled* data during fine-tuning is through an additional stage of language modelling. For this stage, the

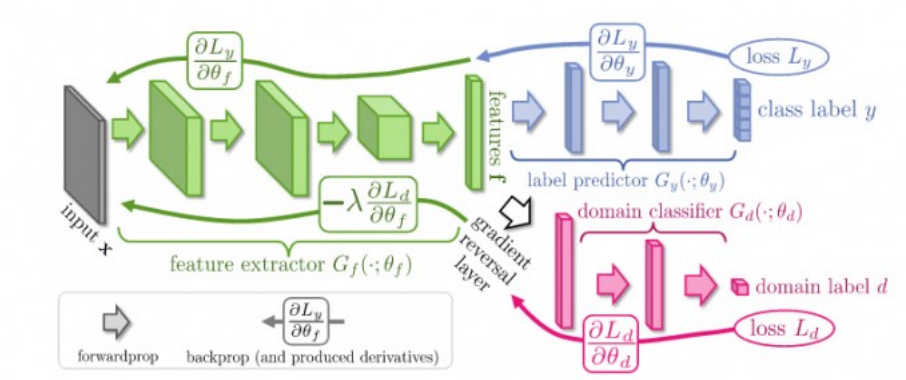


Figure 4.1: The use of the Gradient Reversal Layer in a DNN. Figure from [36].

unlabeled data can either come from the task-specific dataset (i.e. the labels are dropped and language modelling is performed on the input data) [51], or additional unlabeled in-domain corpora [46], [112]. This approach adds a computationally expensive step that requires unlabeled data from a specific source. By contrast, our method leverages out-of-domain unlabeled data with only a small computational overhead and minimal changes to the fine-tuning process.

Our work is also compatible with the semi-supervised learning paradigm [16] that combines learning from both labeled and unlabeled data (section 2.2.3). In this setting, unlabeled data from the task domain are leveraged using a consistency loss which enforces invariance of the output given small peturbations of the input [20], [80]. The adversarial loss term of AFTER can be interpreted as a consistency loss that ensures invariance of representations across domains. However, our method does not require unlabeled data from the same task, that are then augmented through perturbations, but leverages the abundance of unlabeled data from various domains.

4.3 Proposed Approach

In this section we present our approach AFTER, short for domain Adversarial Fine-Tuning as an Effective Regularizer. Figure 4.2 provides a high-level overview of AFTER and Algorithm 1 presents a step by step description of the proposed fine-tuning procedure.

4.3.1 Problem Definition

We tackle a `Main` task, with a labeled dataset from domain $\mathcal{D}_M$, as training data. We further exploit an existing *unlabeled corpus*, `Auxiliary`, that comes from a different domain $\mathcal{D}_{AUX}$. We label each sample of each dataset with the corresponding domain label $y_\mathcal{D}$, $y_\mathcal{D} = 0$ for samples from `Main`, and $y_\mathcal{D} = 1$ for samples from `Auxiliary`. We note that we *do not* use any real labels from `Auxiliary` (if there are any). The domain labels are used to train a classifier that aims to discriminate between $\mathcal{D}_M$ and $\mathcal{D}_{AUX}$.

4.3.2 Model

We initialize our model with pretrained weights from a top-performing language model, such as BERT [28] or XLNET [126]. As presented in section 3.5.3, the representation of both BERT and XLNET for the input sequence is encoded in the `[CLS]` token output

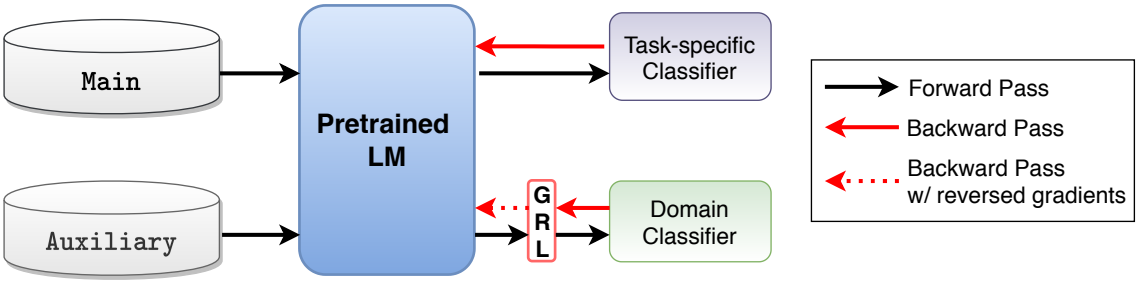


Figure 4.2: Illustration of the proposed approach, AFTER. The task-specific classifier leverages the data from the downstream task (`Main`) while the domain classifier uses unlabeled data from both `Main` and `Auxiliary` datasets as well as the created domain labels.

embedding (in the input representation of BERT this token is located at the beginning of the sequence while in the input representation of XLNET it is located at the end). We add a linear layer on top of the sequence representation (`[CLS]` output embedding) for the `Main` task, resulting in a task-specific loss L_{Main}. We also add another linear layer for the binary domain classifier (Figure 4.2), with a corresponding loss L_{Domain}, which has the same input.

4.3.3 Adversarial Fine-tuning

The domain discriminator outputs a (predicted) domain label for each sample of the training set. We seek representations that are both discriminative for the `Main` task and indiscriminative for the domain classifier. Hence, we minimize L_{Main} and at the same time maximize L_{Domain}, by fine-tuning the pretrained LM with the joint loss:

$$\mathcal{L}_{\text{AFTER}} = L_{Main} - \lambda L_{Domain} \tag{4.1}$$

where λ ($\lambda > 0$) controls the importance of the domain loss. The parameters of the domain classifier are trained to predict the (true) domain label, while the rest of the network is trained to mislead the domain classifier, thereby developing domain-independent internal representations.

4.3.4 Gradient Reversal Layer

We use a Gradient Reversal Layer (`GRL`) [36] between the sequence representation (`[CLS]` output embedding) and the domain discriminator layer, as shown in Figure 4.2, to maximize L_{Domain}. During the forward pass, `GRL` acts as an identity transform, but during backpropagation, `GRL` *reverses* the incoming gradients. We refer the reader to [36] for a more detailed description of the `GRL`. In effect, the pretrained LM parameters are updated towards the opposite direction of the gradient of L_{Main} and, adversarially, towards the direction of the gradient of L_{Domain}.

4.4 Experiments

4.4.1 Datasets

`Main` datasets. We use only four datasets of the GLUE benchmark [120] as `Main` for our experiments, due to resources constraints. The datasets were chosen in such a way to represent the broad variety of natural language understanding tasks (section 3.2), such

Dataset	Domain	N_{train}
`Main`		
CoLA	Miscellaneous	8.5K
SST-2	Movie Reviews	67K
MRPC	News	3.7K
RTE	News, Wikipedia	2.5K
`Auxiliary`		
AG NEWS	Agricultural News (News)	120K
AMAZON	Electronics Reviews (Reviews)	120K
EUROPARL	Legal Documents (Legal)	120K
PUBMED	Medical Papers (Medical)	120K
MATHEMATICS	Mathematics Questions (Math)	120K

Table 4.1: Datasets used. N_{train} denotes the number of training examples. The indicator (domain) summarizes the domain of each `Auxiliary` dataset.

as linguistic acceptability (CoLA) [122], sentiment analysis (SST-2) [108], paraphrase detection (MRPC) [30] and textual entailment (RTE) [5], [10], [25], [38]. The datasets used represent both high (SST-2) and low-resource (RTE, CoLA, MRPC) tasks, as well as single-sentence (CoLA, SST-2) and sentence-pair (MRPC, RTE) tasks.

`Auxiliary` datasets. For the `Auxiliary` data we select corpora from various domains. For the News domain we use the AG NEWS dataset [133] and for the Reviews domain we use a part of the Electronics reviews of He and McAuley [48]. For the Legal domain we use the English part of Europarl [57] and for the Medical domain we use papers from PubMed, provided by Cohan, Dernoncourt, Kim, *et al.* [21]. We also use math questions from the dataset of Saxton, Grefenstette, Hill, *et al.* [100] for the Math domain. Table 4.1 summarizes all datasets. We choose `Auxiliary` datasets that are larger than `Main`, which we consider as the most realistic scenario, given the availability of unlabeled compared to labeled data. We under-sample the `Auxiliary` dataset, as shown in Algorithm 1, to ensure that the two domains are equally represented, motivated by the observation of Bingel and Søgaard [11] that balanced datasets tend to be better in auxiliary tasks.

The `Auxiliary` data are a mixed of labeled and unlabeled datasets. The labeled `Auxiliary` datasets (e.g. AG NEWS) are handled as unlabeled corpora, by dropping the task-specific labels and using only the domain labels. Although some domains might seem identical to those of the `Main` datasets, e.g Electronics Reviews vs. Movie reviews and Agricultural News vs. News this is not the case as can be seen in Figure 4.5.

Preprocessing details. The maximum sequence length for all datasets was 128, so all samples were truncated to 128 tokens and lower-cased. For EUROPARL, which contains parallel corpora in multiple languages, only the English part is used. We therefore sample 120K sentences from the English corpus. For PUBMED we use 120K abstracts from medical papers, from the dataset of Cohan, Dernoncourt, Kim, *et al.* [21]. For MATH we use 120K math questions of medium difficulty from the dataset of Saxton, Grefenstette, Hill, *et al.* [100]. We note that all corpora used are in English.

4.4.2 Implementation Details

We base our implementation on Hugging Face's Transformers library [124] in PyTorch [86]. As our first baseline, we fine-tune the pretrained BERT-BASE model, using the suggested hyperparameters from Devlin, Chang, Lee, *et al.* [28]. Specifically, we use the `bert-base-uncased` model and we fine-tune it using dropout 0.1 and batch size 28. For

the optimization of BERT we use the Adam optimizer [56] without the bias correction and with a learning rate of 2e-5, adam epsilon 1e-6 and weight decay 0.01. We use a linear warmup schedule with 0.1 warmup proportion (see section 2.4.4).

As our next baseline we use the pretrained XLNET-BASE model and we fine-tune it using the suggested hyperparameters from Yang, Dai, Yang, *et al.* [126]. Specifically, we use the `xlnet-base-cased` model (although input sequences were lower-cased) and we fine-tune it using 26 batch size and a learning rate of 2e-5. We also use Adam with bias correction, with the same learning rate and epsilon as BERT (2e-5, 1e-6) but without weight decay or warmup.

When we combine AFTER with either BERT or XLNET we use the same hyperparameters as above. We note that both models have approximately 110M parameters and this is (almost) the same using AFTER, as well. Our approach only introduces a binary domain discriminator in the form of a linear layer. We fine-tune each model for 4 epochs and evaluate the model 5 times per epoch, as suggested by Dodge, Ilharco, Schwartz, *et al.* [29]. We select the best model based on the validation loss.

For each mini-batch, we sample equally from the `Main` and `Auxiliary` datasets, as shown in Algorithm 1. We tune the λ hyperparameter of Eq. 4.1 on the validation set for each experiment, finding that most values of λ improve over the baseline. A more detailed description of λ tuning can be found in section 4.6.4.

Algorithm 1 AFTER algorithm

Choose a pretrained model with parameters $\boldsymbol{\theta}$
procedure ADVERSARIAL FINE-TUNING (`MAIN`, `AUXILIARY`, $\boldsymbol{\theta}, \lambda$)
    Initialize task-specific classifier and domain discriminator parameters randomly: $\boldsymbol{\theta}_{\text{AFTER}} \leftarrow \boldsymbol{\theta} \bigcup \boldsymbol{\theta}_{Main} \bigcup \boldsymbol{\theta}_{Domain}$
    Undersample `Auxiliary` so that $|\texttt{Main}| = |\texttt{Auxiliary}| = N$
    Create domain labels, $y_{\mathcal{D}}$, for both datasets
    while stopping criterion is not met **do**
        for batch in N **do**
            # Create a batch with $\frac{k}{2}$ samples from `Main` and $\frac{k}{2}$ from `Auxiliary`
            Sample $M = (\{x_i, y_i\}, y_{\mathcal{D}i} = 0)_{i=1}^{\frac{k}{2}} \in \texttt{Main}$
            Sample $A = (\{x_j\}, y_{\mathcal{D}j} = 1)_{j=1}^{\frac{k}{2}} \in \texttt{Auxiliary}$
            $batch \leftarrow M \bigcup A$
            for i in $batch$ **do**
                if $x_i \in M$ **then**
                    Compute $L_{Main}(\hat{y}_i, y_i; \boldsymbol{\theta}_{\text{AFTER}})$
                Compute $L_{Domain}(\hat{y}_{\mathcal{D}i}, y_{\mathcal{D}i}; \boldsymbol{\theta}_{\text{AFTER}})$
            # Compute joint loss
            $\mathcal{L}_{\text{AFTER}} = \sum_{i=1}^{\frac{k}{2}} L_{Main}(\hat{y}_i, y_i; \boldsymbol{\theta}_{\text{AFTER}}) - \lambda \sum_{i=1}^{k} L_{Domain}(\hat{y}_{\mathcal{D}i}, y_{\mathcal{D}i}; \boldsymbol{\theta}_{\text{AFTER}})$
            Update model $\boldsymbol{\theta}_{\text{AFTER}} \leftarrow \boldsymbol{\theta}_{\text{AFTER}} - \epsilon \nabla_{\boldsymbol{\theta}_{\text{AFTER}}} \mathcal{L}_{\text{AFTER}}$

4.5 Results

Table 4.2 shows the results on the validation sets of the four GLUE datasets for the two pretrained LMs. We do not report results on CoLA with XLNET (−) because the model did not converge using the chosen hyperparameters. In order to replicate the results

	CoLA	SST-2	MRPC	RTE
	Matthews corr.	*Accuracy*	*Accuracy / F1*	*Accuracy*
BERT	55.5 ± 3.2	92.0 ± 0.5	85.4 ± 1.1 / 89.6 ± 0.6	64.3 ± 3.1
AFTER W/ NEWS	$\mathbf{57.3} \pm 1.5$	$\underline{92.5} \pm 0.4$	$\mathbf{87.5} \pm 1.7$ / $\mathbf{91.1} \pm 1.2$	$\underline{64.7} \pm 1.9$
AFTER W/ REVIEWS	$\underline{57.1} \pm 1.2$	$\underline{92.4} \pm 0.3$	$\underline{86.4} \pm 0.3$ / $\underline{90.1} \pm 0.4$	$\underline{64.6} \pm 0.8$
AFTER W/ LEGAL	55.0 ± 1.5	$\underline{92.4} \pm 0.3$	$\underline{86.6} \pm 0.6$ / $\underline{90.3} \pm 0.5$	$\mathbf{64.8} \pm 1.9$
AFTER W/ MEDICAL	$\underline{55.9} \pm 2.9$	$\mathbf{92.6} \pm 0.3$	$\underline{86.9} \pm 1.3$ / $\underline{90.7} \pm 1.0$	62.6 ± 3.4
AFTER W/ MATH	$\underline{56.1} \pm 2.8$	$\underline{92.3} \pm 0.8$	$\underline{87.3} \pm 0.9$ / $\underline{90.8} \pm 0.7$	62.5 ± 1.3
XLNET	–	93.0 ± 0.7	86.4 ± 0.7 / 90.1 ± 0.5	64.7 ± 4.4
AFTER W/ NEWS	–	$\mathbf{93.9} \pm 0.3$	$\underline{87.3} \pm 0.7$ / $\underline{91.0} \pm 0.5$	63.9 ± 2.3
AFTER W/ REVIEWS	–	$\underline{93.5} \pm 0.3$	$\underline{86.9} \pm 0.6$ / $\underline{90.5} \pm 0.5$	$\underline{65.1} \pm 2.8$
AFTER W/ LEGAL	–	$\underline{93.6} \pm 0.5$	$\mathbf{87.5} \pm 1.6$ / $\mathbf{90.9} \pm 1.2$	$\underline{64.8} \pm 1.6$
AFTER W/ MEDICAL	–	$\underline{93.3} \pm 0.5$	$\underline{87.0} \pm 1.1$ / 90.5 ± 0.7	64.5 ± 2.1
AFTER W/ MATH	–	$\mathbf{93.9} \pm 0.4$	$\underline{87.3} \pm 1.2$ / $\underline{90.8} \pm 0.9$	$\mathbf{66.1} \pm 1.9$

Table 4.2: Comparison of standard of fine-tuning and AFTER for BERT (Top) and XLNET (Bottom). Underlined scores outperform the baseline. Best scores are shown in **bold**. We report the mean and standard deviation across five runs on the validation set.

of Yang, Dai, Yang, *et al.* [126], the authors suggested using a considerably larger batch size ($\times 4$), which was not possible in our case, due to resources constraints[1].

4.5.1 Evaluation on BERT

We observe in Table 4.2 that the proposed approach (AFTER) outperforms the first baseline (BERT) in all four tasks. For most of these tasks, AFTER results in improved performance with every `Auxiliary` dataset, demonstrating the robustness of our approach across domains.

Specifically, in CoLA, we observe that fine-tuning with the adversarial loss substantially outperforms standard BERT fine-tuning. We observe that AFTER improves the baseline by 1.8 points using an `Auxiliary` dataset from the NEWS domain, while the use of `Auxiliary` datasets from other domains also improve performance and reduce variance. In SST-2, we notice that although BERT achieves high accuracy, the use of AFTER still results in slight performance gains ($\sim 0.4\%$). Similar to CoLA, these improvements are consistent across `Auxiliary` datasets and often come with a reduced variance, compared to BERT. For instance, using an `Auxiliary` dataset from the MEDICAL domain we improve the accuracy of BERT from 92.0% to 92.6%. In MRPC, we observe gains of 1.5 points on average in accuracy and 1.0 in F1 over BERT. Using NEWS data as `Auxiliary`, AFTER outperforms the baseline by 2.1 points in accuracy and 1.5 in F1. In RTE, the proposed approach improves upon the baseline from 64.3% to 64.8% in accuracy, using data from the LEGAL domain. However, we also observe deteriorated performance with the use of some `Auxiliary` datasets (e.g. MEDICAL, MATH). We attribute this result to the similarity between the domain of RTE (Wikipedia) and the domain of the pretraining corpus of BERT (Wikipedia and Books). We test this hypothesis in section 4.6.

[1]The authors' response regarding the hyperaparameters : Clarification on reported dev numbers on GLUE tasks. Similar problems have been reported for other BERT-based models on CoLA, as well: Xlnet, Alberta, Roberta are not finetuned for CoLA task.

4.5.2 Evaluation on XLNet

To test the generality of the proposed fine-tuning method, we apply AFTER to XLNet [126], another powerful pretrained LM. We conduct experiments on three of the four considered datasets, due to resources constraints. As presented in section 3.5, XLNet has a similar architecture as BERT and uses a different pretraining objective that results in improved performance. We observe in Table 4.2 that AFTER results in performance boost for an even higher-performing LM baseline.

Specifically, in SST-2 AFTER improves the accuracy of standard XLNet fine-tuning by 0.6% on average and reduces variance, as well. For instance, with the use of `Auxiliary` data from News or Math domains, AFTER results in 0.9% improvement in accuracy. In MRPC, the performance boost is consistent across `Auxiliary` data and the use of Legal data leads in absolute improvement of 1.1% in accuracy and 0.8% in F1. In RTE, adversarial fine-tuning outperforms the baseline by 1.4% in accuracy. However, similar to BERT, we observe worse performance when using AFTER with some `Auxiliary` data (e.g. News, Medical). We attribute this performance degradation to the same reason as BERT, the similarity between the pretraining corpus domain and the target task domain.

4.5.3 Summary

The experiments of this section reveal that using AFTER compared to standard fine-tuning can boost target task performance while also reduce variance. We can therefore attribute the effectiveness of AFTER to regularization itself and not to the model architecture. In addition, combining the results from the two sets of experiments, we observe that AFTER, combined with a top-performing pretrained LM (BERT+AFTER), matches (RTE) or surpasses (MRPC) the gains observed by using a higher-performing LM (XLNet). This finding demonstrates the effectiveness of the proposed adversarial fine-tuning approach and motivates the need for more effective fine-tuning schemes as a way to improve the target task performance of pretrained LMs.

4.6 Ablation Study

In this section, we investigate the effect of some key factors of AFTER such as the relation of the target task domain and the domain of the pretraining corpus of the LM, the selection of `Auxiliary` data, the emergence of domain-invariant characteristics and the choice of λ.

4.6.1 LM pretraining and Task Domains

To explore why AFTER fails to improve upon the baselines on some experiments in RTE, we examine if the pretrained representations are already well suited for the task (i.e. no regularization is needed). We, thus, calculate the average masked language modelling (MLM) loss of BERT for each `Main` dataset (the same applies to XLNet since it is pretrained on the same data) by performing an inference step with the MLM objective. We observe in Table 4.3 that SST-2 produces the largest loss (3.39), which can be partially attributed to the dataset format (it contains short sentences that make the MLM task very challenging) and can be excluded from this analysis. RTE produces the lowest loss (2.17), confirming our intuition regarding the similarity of the pretraining corpus of BERT and RTE. In this case, general-domain (created during pretraining) and domain-specific representations

(created during fine-tuning) are close, rendering domain-adversarial regularization undesirable. This is also confirmed by the the vocabulary overlap (see section 4.6.2) between RTE and a Wikipedia corpus (Table 4.3). The more distant the pretraining domain of the pretrained LM is to the specific task (measured by vocabulary overlap and MLM loss), the more benefits AFTER demonstrates, confirming our intuition regarding domain-adversarial regularization.

	RTE	**MRPC**	**CoLA**	**SST-2**
MLM Loss	**2.17**	2.37	2.53	3.39
Overlap with WIKI (%)	**38.3**	34.0	24.0	26.1
AFTER Improvement (%)	**0.8**	2.5	3.2	0.7

Table 4.3: Masked LM loss of BERT (the lower the better), vocabulary overlap with the Wikipedia domain (WIKI) and relative improvement of AFTER (best) for each task.

4.6.2 Domain Distance

We measure the domain distance for all `Main-Auxiliary` pairs to evaluate how the choice of the latter affects the performance of AFTER. We represent the word distribution of each `Main` and `Auxiliary` dataset using term distributions: $t \in \mathcal{R}^{|V|}$ where t_i is the probability of the i-th word in the joint vocabulary V [91]. We employ Jensen-Shannon (JS) divergence as the distance measure. In order to create a common vocabulary V for all data we find the $5k$ most frequent words in each dataset and we then take the union of these sub-vocabularies which results in $23k$ words. Combining the results of Table 4.2 and Fig. 4.3, no clear pattern emerges demonstrating, perhaps, our method's robustness to domain distance.

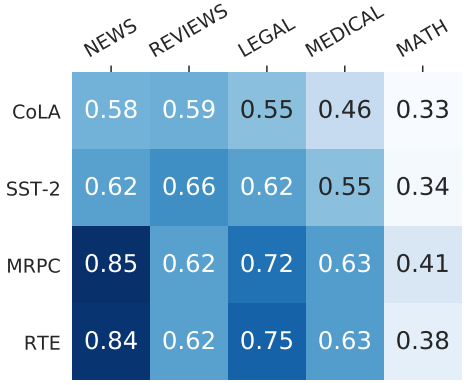


Figure 4.3: JS divergence between all dataset pairs.

We also conduct a similar analysis with vocabulary overlap, by creating each domain (or task) vocabulary with the $10k$ most frequent words in each dataset (in case a dataset contains less words we use all the words in the dataset). First, we calculate the vocabulary overlap between domains (Figure 4.4). We observe in Figure 4.4 that most domains are dissimilar, with the exception of NEWS and LEGAL domains, that have 36.6% vocabulary overlap. We also observe that the MATH domain has the most unique vocabulary, compared with the other domains, something that is expected due to the presence of many numbers and mathematical symbols.

We subsequently calculate the vocabulary overlap between each task and all domains (Figure 4.5). For this purpose, we also include the WIKI domain to account for the pre-

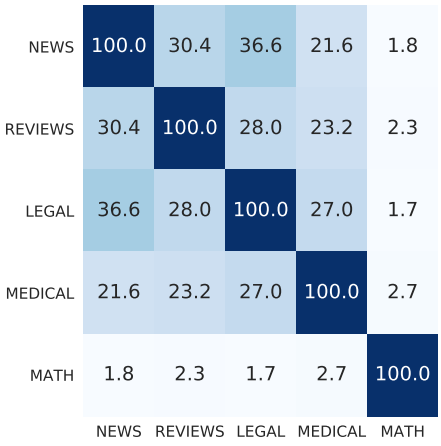


Figure 4.4: Vocabulary overlap (%) between domains.

training domain of BERT and XLNet. For the vocabulary of Wiki we use the WikiText-2 corpus from Merity, Xiong, Bradbury, *et al.* [75]. In Figure 4.5, we observe that RTE has the most overlapping vocabulary with Wiki which is a possible cause for the deteriorated performance of after, since the model has already been pretrained in this domain and does not require further regularization, as described in section 4.6.1.

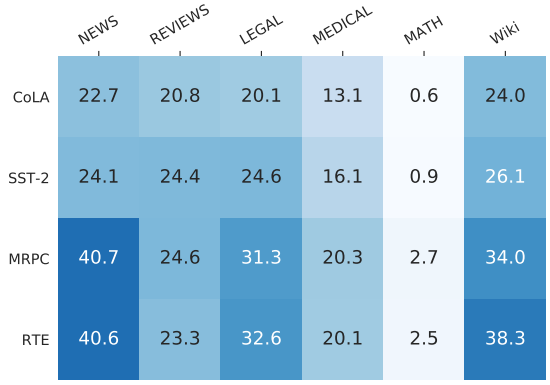


Figure 4.5: Vocabulary overlap (%) between tasks and domains.

Both methods did not reveal a clear pattern regarding the effect of the domain of `Auxiliary` data on the performance of after. We leave a further investigation of selection criteria for the `Auxiliary` data for future work.

4.6.3 Domain-invariant vs. Domain-specific Features

To investigate if the benefits of after can be attributed only to data augmentation we compare adversarial ($\lambda > 0$ in Eq. 4.1) and multi-task ($\lambda < 0$) fine-tuning. We therefore fine-tune BERT on MRPC and CoLA for both settings and we tune each λ separately for each setting for fair comparison. We observe that during multi-task fine-tuning (Fig. 4.6), L_{Domain} is close to zero (even in the first epoch). This implies that domain classification is an easy auxiliary task, confirming our intuition that a non-adversarial fine-tuning setting favors domain-specific features. Although the multi-task approach leverages the same unlabeled data, its performance is worse than after (Table 4.4), which highlights the need for an adversarial domain discriminator. The main takeaway from this set of

experiments is that the preservation of domain-invariant features with AFTER results in better performance.

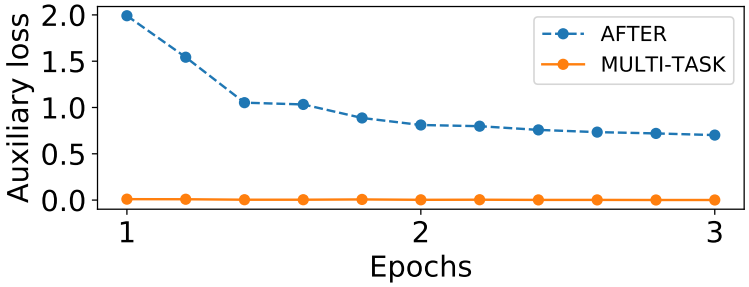


Figure 4.6: AFTER ($\lambda > 0$) vs. MULTI-TASK ($\lambda < 0$).

	CoLA	MRPC
AFTER W/ NEWS	**57.3**	**87.5/91.1**
MULTI-TASK W/ NEWS	56.5	86.7/90.5

Table 4.4: Comparison of AFTER vs. MULTI-TASK. Results are averages across 5 runs.

4.6.4 Tuning the λ hyperparameter

We tune λ on each development set, choosing from $\{0.1, 0.01, 0.001, 0.0001\}$. In Figure 4.7 we compare the performance of BERT and AFTER for different `Main-Auxiliary` combinations, as we vary the value of λ, across three runs.

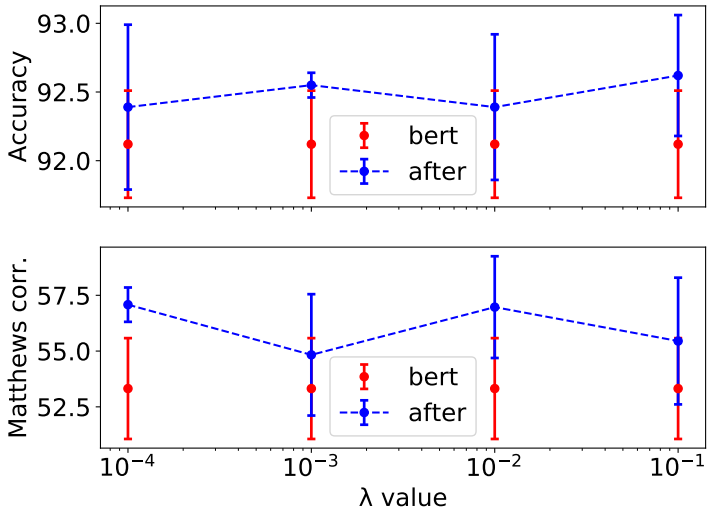


Figure 4.7: Performance of standard BERT fine-tuning vs. AFTER for different λ values on SST-2 (Top) and CoLA (Bottom). The errorbars correspond to one standard deviation.

We observe that the various values of λ can have different effect on the performance and variance of AFTER. We observe that most values of λ significantly improve the performance of the baseline, BERT and an exhaustive search is not required. Table 4.5 presents the values of λ that were used for the results reported in Table 4.2, for both LMs. Best values of λ were chosen based on the task-specific metric (e.g. Accuracy, Matthews correlation).

	BERT				XLNet		
	CoLA	**SST-2**	**MRPC**	**RTE**	**SST-2**	**MRPC**	**RTE**
News	0.1	0.1	0.1	0.001	0.01	0.1	0.0001
Reviews	0.1	0.01	0.1	0.001	0.001	0.0001	0.01
Legal	0.01	0.1	0.01	0.1	0.001	0.01	0.0001
Medical	0.01	0.1	0.1	0.0001	0.1	0.1	0.01
Math	0.001	0.1	0.001	0.001	0.0001	0.01	0.01

Table 4.5: Best λ value of AFTER for each experiment.

4.7 Conclusions and Future Work

We propose AFTER, a domain adversarial method to regularize the fine-tuning process of a pretrained LM. Empirical results demonstrate that our method can lead to improved performance over standard fine-tuning for two pretrained LMs. AFTER can be widely applied to any transfer learning setting and model architecture, with minimal changes to the fine-tuning process, without requiring any additional labeled data. We aim to further explore the effect of `Auxiliary` data on the final performance and the use of multiple `Auxiliary` datasets. We also aim to extend the proposed approach as a way to fine-tune a pretrained LM to a different language, in order to produce language-invariant representations.

Chapter 5

Conclusions

In this work, we investigate alternative ways of transferring pretrained language models in order to solve text classification tasks. We propose a fine-tuning approach that introduces a novel regularizing method, in addition to the task-specific classification objective. The proposed fine-tuning approach (AFTER) leverages only unlabeled data (in addition to the task-specific dataset) and improves performance on downstream tasks. Our goal is to preserve the knowledge gained during the pretraining of the language model while adapting to the particular task.

The capabilities of pretrained language models range from state-of-the-art results [68], [69], [126] when fine-tuned to NLP benchmarks [120] to zero-shot performance in tasks they have not been explicitly trained on [95], [96]. We argue that part of this knowledge is lost during the fine-tuning process of LMs and we tackle this issue by proposing domain adversarial fine-tuning of pretrained LMs.

The text classification tasks examined in this work are of great practical importance, especially sentiment analysis and textual entailment. Systems that are able to classify the sentiment of a text or identify entailment between pieces of text could have numerous applications in real-world scenarios. While in the context of this work, we focus on NLP and pretrained LMs, the proposed method could be equally applied to other ML fields such as computer vision or speech processing. The scenario that we examine resembles the common transfer learning scheme were a model, pretrained on a large dataset, is transferred to a task with a much smaller dataset.

5.1 Discussion

Recently, most NLP tasks are addressed by leveraging a language model, pretrained on large corpus, that is fine-tuned to the task-specific dataset [28], [51], [96]. We argue that this process can result in loss of the knowledge capturing during the pretraining of LM, a phenomenon called catastrophic forgetting [43]. In order to effectively exploit the capabilities of pretrained LMs and improve the overall transferring process we introduce an adversarial fine-tuning method, AFTER. The proposed method leverages unlabeled data that come from different domains and tasks, in order to regularize the fine-tuning process and preserve the knowledge gained during pretrainng. Specifically, for each task we want to solve, we select an unlabeled corpus that comes from a different domain. Then, during fine-tuning we constrain the degree to which the representations of the model for the two datasets are allowed to differ. By adding this extra loss term we prevent the emergence of too domain specific feature that could hurt the generalization ability of the model and lead to overall worse performance.

We evaluate our method on four text classification tasks: sentiment analysis, linguistic acceptability, paraphrase detection and textual entailment. The proposed approach outperforms the standard fine-tuning technique on most of the settings examined, demonstrating another possible direction of research regarding transferring techniques for pretrained models. Our fine-tuning method is easy to implement since it requires minimal changes on the fine-tuning process and leverages additional unlabeled data.

5.2 Limitations and Future work

A possible limitation of our work is the marginal gains on some experiments, compared to the standard fine-tuning method. This can be attributed to the state-of-the-art performance of the pretrained LM which does not leave much room for improvement. However, as empirical results suggest, our method combined with a high-performing LM can outweigh the benefits of using a higher-performing LM on some tasks. The improvements in performance by our approach could be more clearly demonstrated in a simpler LM architecture, such as LSTM-based models. We intend to explore the effectiveness of our method to such models.

An inherent drawback of the proposed fine-tuning approach is the fact that it doubles the fine-tuning time. Our method effectively doubles the size of the task-specific dataset (it is augmented with the out-of-domain unlabeled data) and we, therefore, need to perform more optimization steps. We argue that the increased performance and reduced variance could compensate for the extended fine-tuning time. In contrast to other variants of the fine-tuning process that either require additional fine-tuning on a data-rich task [90] or performing a computationally costly language modelling step [46], [51], [112], our method is integrated in the standard fine-tuning process (i.e. does not introduce an extra step) and only doubles the fine-tuning time, which in the case of small datasets can be a negligible difference.

Another limitation could be the choice of the λ hyperparameter that controls the importance of the second loss term. However, we empirically found that it does not require extensive tuning and it is possible to find a value of this hyperparameter that can work reasonably well for many settings, which is the case for this work.

In addition, our ablation study did not reveal any evident correlation between the distance between the two domains and the effectiveness of our method. We aim to further explore the effect of the `Auxiliary` dataset on the performance of the proposed approach. Apart from a more extensive analysis regarding the similarity of the domains, another interesting research direction would be to investigate how the `Main` / `Auxiliary` ratio affects performance. Similar analysis on data augmentation methods [33], [129] have revealed interesting patterns.

Finally, in addition to a simpler, baseline LM we aim to experiment with more current state-of-the-art pretrained LMs [31], [62], [69] and apply the domain adversarial fine-tuning method to different tasks, domains and even languages.

Abbreviations

AFTER	domain Adversarial Fine-Tuning as an Effective Regularizer
AI	Artificial Intelligence
ANN	Artificial Neural Network
BERT	Bidirectional Encoder Representations from Transformers
BN	Batch Normalization
BoW	Bag-of-Words
CBOW	Continuous Bag-of-Words
CLM	Conditional/Causal Language Modelling
DL	Deep Learning
DNN	Deep Neural Network
ELMo	Embeddings from Language Models
FFNN	Feed-Forward Neural Network
FN	False Negative
FP	False Positive
GELU	Gaussian Error Linear Unit
GRL	Gradient Reversal Layer
JS	Jensen–Shannon
LM	Language Model
LN	Layer Normalization
LSTM	Long Short-Term Memory
MCC	Matthews Correlation Coefficient
ML	Machine Learning
MLM	Masked Language Modelling
MLP	Multilayer Perceptron
MTL	Multi-task Learning
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
NSP	Next Sentence Prediction
OOV	Out-of-Vocabulary
PLM	Permutation Language Modelling
POS	Part-of-Speech
PPMI	Positive Pointwise Mutual Information
QA	Question Answering
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
TF-IDF	Term Frequency-Inverse Document Frequency
TN	True Negative
TP	True Positive
ULMFiT	Universal Language Model Fine-Tuning

Bibliography

[1] E. Akgün and M. Demir, «Modeling Course Achievements of Elementary Education Teacher Candidates with Artificial Neural Networks», *International Journal of Assessment Tools in Education*, vol. 5, 2018. DOI: 10.21449/ijate.444073.

[2] R. Al-Rfou, D. Choe, N. Constant, M. Guo, and L. Jones, «Character-Level Language Modeling with Deeper Self-Attention», in *AAAI*, 2018.

[3] J. Ba, J. Kiros, and G. Hinton, «Layer Normalization», 2016.

[4] D. Bahdanau, K. Cho, and Y. Bengio, «Neural Machine Translation by Jointly Learning to Align and Translate», in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.0473.

[5] R. Bar-Haim, I. Dagan, B. Dolan, L. Ferro, D. Giampiccolo, and B. Magnini, «The Second PASCAL Recognising Textual Entailment Challenge», 2006.

[6] M. Baroni, G. Dinu, and G. Kruszewski, «Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors», in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014, pp. 238–247. DOI: 10.3115/v1/P14-1023. [Online]. Available: https://www.aclweb.org/anthology/P14-1023.

[7] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, «Analysis of Representations for Domain Adaptation», in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds., 2007, pp. 137–144. [Online]. Available: http://papers.nips.cc/paper/2983-analysis-of-representations-for-domain-adaptation.pdf.

[8] Y. Bengio, P. Simard, and P. Frasconi, «Learning Long-Term Dependencies with Gradient Descent is Difficult», *Trans. Neur. Netw.*, vol. 5, no. 2, 157–166, 1994, ISSN: 1045-9227. DOI: 10.1109/72.279181. [Online]. Available: https://doi.org/10.1109/72.279181.

[9] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, «A Neural Probabilistic Language Model», *J. Mach. Learn. Res.*, vol. 3, no. null, 1137–1155, 2003, ISSN: 1532-4435.

[10] L. Bentivogli, I. Dagan, H. T. Dang, D. Giampiccolo, and B. Magnini, «The Fifth PASCAL Recognizing Textual Entailment Challenge», in *Proceedings of the Text Analysis Conference (TAC)*, 2009.

[11] J. Bingel and A. Søgaard, «Identifying beneficial task relations for multi-task learning in deep neural networks», in *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics*, 2017, pp. 164–169. [Online]. Available: https://www.aclweb.org/anthology/E17-2026/.

[12] J. Blitzer, M. Dredze, and F. Pereira, «Biographies, Bollywood, Boom-boxes and Blenders: Domain Adaptation for Sentiment Classification», in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, 2007, pp. 440–447. [Online]. Available: https://www.aclweb.org/anthology/P07-1056.

[13] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, «Enriching Word Vectors with Subword Information», *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017. DOI: 10.1162/tacl_a_00051. [Online]. Available: https://www.aclweb.org/anthology/Q17-1010.

[14] R. Caruana, «Multitask Learning», in *Learning to Learn*. 1998, 95–133, ISBN: 0792380479.

[15] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, B. Strope, and R. Kurzweil, «Universal Sentence Encoder for English», in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018.

[16] O. Chapelle, B. Schlkopf, and A. Zien, *Semi-Supervised Learning*, 1st. 2010, ISBN: 0262514125.

[17] Z. Chen, R. Yang, Z. Zhao, D. Cai, and X. He, «Dialogue Act Recognition via CRF-Attentive Structured Network», *CoRR*, vol. abs/1711.05568, 2017. arXiv: 1711.05568. [Online]. Available: http://arxiv.org/abs/1711.05568.

[18] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, «Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation», in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. [Online]. Available: https://www.aclweb.org/anthology/D14-1179.

[19] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, «What Does BERT Look at? An Analysis of BERT's Attention», in *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 2019, pp. 276–286. DOI: 10.18653/v1/W19-4828. [Online]. Available: https://www.aclweb.org/anthology/W19-4828.

[20] K. Clark, M.-T. Luong, C. D. Manning, and Q. Le, «Semi-Supervised Sequence Modeling with Cross-View Training», in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 1914–1925. DOI: 10.18653/v1/D18-1217. [Online]. Available: https://www.aclweb.org/anthology/D18-1217.

[21] A. Cohan, F. Dernoncourt, D. S. Kim, T. Bui, S. Kim, W. Chang, and N. Goharian, «A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents», in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 2018, pp. 615–621. DOI: 10.18653/v1/N18-2097. [Online]. Available: https://www.aclweb.org/anthology/N18-2097.

[22] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, «Supervised Learning of Universal Sentence Representations from Natural Language Inference Data», *CoRR*, vol. abs/1705.02364, 2017. arXiv: 1705.02364. [Online]. Available: http://arxiv.org/abs/1705.02364.

[23] A. Conneau and G. Lample, «Cross-lingual Language Model Pretraining», in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d´ Alché-Buc, E. Fox, and R. Garnett, Eds., 2019, pp. 7059–7069. [Online]. Available: http://papers.nips.cc/paper/8928-cross-lingual-language-model-pretraining.pdf.

[24] G. Cybenko, «Approximation by superpositions of a sigmoidal function», *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989. DOI: 10.1007/BF02551274. [Online]. Available: https://doi.org/10.1007/BF02551274.

[25] I. Dagan, O. Glickman, and B. Magnini, «The PASCAL Recognising Textual Entailment Challenge», in *Proceedings of the First International Conference on Machine Learning Challenges: Evaluating Predictive Uncertainty Visual Object Classification, and Recognizing Textual Entailment*, ser. MLCW'05, Southampton, UK, 2005, 177–190, ISBN: 3540334270. DOI: 10.1007/11736790_9. [Online]. Available: https://doi.org/10.1007/11736790_9.

[26] A. M. Dai and Q. V. Le, «Semi-supervised Sequence Learning», in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 3079–3087. [Online]. Available: http://papers.nips.cc/paper/5949-semi-supervised-sequence-learning.pdf.

[27] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. Le, and R. Salakhutdinov, «Transformer-XL: Attentive Language Models beyond a Fixed-Length Context», in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 2978–2988. DOI: 10.18653/v1/P19-1285. [Online]. Available: https://www.aclweb.org/anthology/P19-1285.

[28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding», in *In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019, pp. 4171–4186. [Online]. Available: https://www.aclweb.org/anthology/N19-1423/.

[29] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi, and N. A. Smith, «Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping», *ArXiv*, vol. abs/2002.06305, 2020.

[30] W. B. Dolan and C. Brockett, «Automatically Constructing a Corpus of Sentential Paraphrases», in *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005. [Online]. Available: https://www.aclweb.org/anthology/I05-5002.

[31] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H.-W. Hon, «Unified Language Model Pre-training for Natural Language Understanding and Generation», in *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019.

[32] J. Duchi, E. Hazan, and Y. Singer, «Adaptive subgradient methods for online learning and stochastic optimization», *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[33] S. Edunov, M. Ott, M. Auli, and D. Grangier, «Understanding Back-Translation at Scale», in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 489–500. DOI: 10.18653/v1/D18-1045. [Online]. Available: https://www.aclweb.org/anthology/D18-1045.

[34] D. A. Ferrucci, «IBM's Watson/DeepQA», *SIGARCH Comput. Archit. News*, vol. 39, no. 3, 2011, ISSN: 0163-5964. DOI: 10.1145/2024723.2019525. [Online]. Available: https://doi.org/10.1145/2024723.2019525.

[35] Y. Gal and Z. Ghahramani, «A Theoretically Grounded Application of Dropout in Recurrent Neural Networks», in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., 2016, pp. 1019–1027. [Online]. Available: http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-recurrent-neural-networks.pdf.

[36] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, «Domain-adversarial Training of Neural Networks», *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016. [Online]. Available: http://jmlr.org/papers/volume17/15-239/15-239.pdf.

[37] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, «Convolutional Sequence to Sequence Learning», in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, 2017, pp. 1243–1252. [Online]. Available: http://proceedings.mlr.press/v70/gehring17a.html.

[38] D. Giampiccolo, B. Magnini, I. Dagan, and B. Dolan, «The Third PASCAL Recognizing Textual Entailment Challenge», in *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, 2007, pp. 1–9. [Online]. Available: https://www.aclweb.org/anthology/W07-1401.

[39] X. Glorot, A. Bordes, and Y. Bengio, «Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach», 2011.

[40] X. Glorot, A. Bordes, and Y. Bengio, «Deep Sparse Rectifier Neural Networks», in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., ser. Proceedings of Machine Learning Research, vol. 15, 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html.

[41] B. Gong, K. Grauman, and F. Sha, «Connecting the Dots with Landmarks: Discriminatively Learning Domain-Invariant Features for Unsupervised Domain Adaptation», in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., ser. Proceedings of Machine Learning Research, vol. 28, 2013, pp. 222–230. [Online]. Available: http://proceedings.mlr.press/v28/gong13.html.

[42] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. 2016, http://www.deeplearningbook.org.

[43] I. J. Goodfellow, M. Mirza, X. Da, A. C. Courville, and Y. Bengio, «An Empirical Investigation of Catastrophic Forgeting in Gradient-Based Neural Networks», *CoRR*, vol. abs/1312.6211, 2013. [Online]. Available: https://arxiv.org/abs/1312.6211.

[44] I. J. Goodfellow, J. Shlens, and C. Szegedy, «Explaining and Harnessing Adversarial Examples», in *In Proceedings of the International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.6572.

[45] K. Grauman, «Geodesic Flow Kernel for Unsupervised Domain Adaptation», in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2012, pp. 2066–2073. [Online]. Available: http://dl.acm.org/citation.cfm?id=2354409.2355024.

[46] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith, «Don't Stop Pretraining: Adapt Language Models to Domains and Tasks», *ArXiv*, vol. abs/2004.10964, 2020.

[47] K. He, X. Zhang, S. Ren, and J. Sun, «Deep Residual Learning for Image Recognition», *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[48] R. He and J. McAuley, «Ups and Downs: Modeling the Visual Evolution of Fashion Trends with One-Class Collaborative Filtering», in *Proceedings of the International Conference on World Wide Web*, ser. WWW '16, Montréal, Québec, Canada, 2016, pp. 507–517, ISBN: 978-1-4503-4143-1. DOI: 10.1145/2872427.2883037. [Online]. Available: https://doi.org/10.1145/2872427.2883037.

[49] D. Hendrycks and K. Gimpel, «Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units», *CoRR*, vol. abs/1606.08415, 2016. arXiv: 1606.08415. [Online]. Available: http://arxiv.org/abs/1606.08415.

[50] S. Hochreiter and J. Schmidhuber, «Long Short-Term Memory», *Neural Comput.*, vol. 9, no. 8, 1735–1780, 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

[51] J. Howard and S. Ruder, «Universal Language Model Fine-tuning for Text Classification», in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2018, pp. 328–339. [Online]. Available: https://www.aclweb.org/anthology/P18-1031.

[52] S. Ioffe and C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift», in *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html.

[53] R. Johnson and T. Zhang, «Deep Pyramid Convolutional Neural Networks for Text Categorization», in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 562–570. DOI: 10.18653/v1/P17-1052. [Online]. Available: https://www.aclweb.org/anthology/P17-1052.

[54] A. Karpathy, «Convolutional Neural Networks for Visual Recognition», [Online]. Available: http://cs231n.github.io/neural-networks-1/.

[55] P. Keung, Y. Lu, and V. Bhardwaj, «Adversarial Learning with Contextual Embeddings for Zero-resource Cross-lingual Classification and NER», in *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2019, pp. 1355–1360. [Online]. Available: https://www.aclweb.org/anthology/D19-1138.

[56] D. P. Kingma and J. Ba, «Adam: A Method for Stochastic Optimization», in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980.

[57] P. Koehn, «EuroParl: A parallel corpus for statistical machine translation», vol. 5, 2004.

[58] A. Krizhevsky, I. Sutskever, and G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks», in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., 2012, pp. 1097–1105. [Online]. Available: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[59] T. Kudo and J. Richardson, «SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing», in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2018, pp. 66–71. DOI: `10.18653/v1/D18-2012`. [Online]. Available: `https://www.aclweb.org/anthology/D18-2012`.

[60] G. Lample, A. Conneau, L. Denoyer, and M. Ranzato, «Unsupervised Machine Translation Using Monolingual Corpora Only», in *Proceedings of the International Conference on Learning Representations*, 2018. [Online]. Available: `https://openreview.net/forum?id=rkYTTf-AZ`.

[61] G. Lample, A. Conneau, M. Ranzato, L. Denoyer, and H. Jégou, «Word translation without parallel data», in *Proceedings of the International Conference on Learning Representations*, 2018. [Online]. Available: `https://openreview.net/forum?id=H196sainb`.

[62] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, «ALBERT: A Lite BERT for Self-supervised Learning of Language Representations», in *International Conference on Learning Representations*, 2020. [Online]. Available: `https://openreview.net/forum?id=H1eA7AEtvS`.

[63] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng, «Building High-Level Features Using Large Scale Unsupervised Learning», in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ser. ICML'12, Edinburgh, Scotland, 2012, 507–514, ISBN: 9781450312851.

[64] Y. LeCun, Y. Bengio, and G. Hinton, «Deep Learning», *Nature*, vol. 521, pp. 436–44, 2015. DOI: `10.1038/nature14539`.

[65] S. Lee, D. Kim, and J. Park, *Domain-agnostic Question-Answering with Adversarial Training*, 2019. arXiv: `1910.09342 [cs.CL]`.

[66] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, «Multilayer feedforward networks with a nonpolynomial activation function can approximate any function», *Neural Networks*, vol. 6, no. 6, pp. 861 –867, 1993, ISSN: 0893-6080. DOI: `https://doi.org/10.1016/S0893-6080(05)80131-5`. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0893608005801315`.

[67] Z. Lin, M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, «A Structured Self-attentive Sentence Embedding», 2017. arXiv: `1703.03130 [cs.CL]`.

[68] X. Liu, P. He, W. Chen, and J. Gao, «Multi-Task Deep Neural Networks for Natural Language Understanding», in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 4487–4496. DOI: `10.18653/v1/P19-1441`. [Online]. Available: `https://www.aclweb.org/anthology/P19-1441`.

[69] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, «RoBERTa: A Robustly Optimized BERT Pretraining Approach», *CoRR*, vol. abs/1907.11692, 2019. arXiv: 1907.11692. [Online]. Available: http://arxiv.org/abs/1907.11692.

[70] M. Long and J. Wang, «Learning Transferable Features with Deep Adaptation Networks», *CoRR*, vol. abs/1502.02791, 2015. arXiv: 1502.02791. [Online]. Available: http://arxiv.org/abs/1502.02791.

[71] M.-T. Luong, H. Pham, and C. D. Manning, «Effective Approaches to Attention-based Neural Machine Translation», 2015. arXiv: 1508.04025 [cs.CL].

[72] B. McCann, J. Bradbury, C. Xiong, and R. Socher, «Learned in Translation: Contextualized Word Vectors», in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., 2017, pp. 6294–6305. [Online]. Available: http://papers.nips.cc/paper/7209-learned-in-translation-contextualized-word-vectors.pdf.

[73] W. S. McCulloch and W. Pitts, «A logical calculus of the ideas immanent in nervous activity», *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[74] S. Merity, N. S. Keskar, and R. Socher, «Regularizing and Optimizing LSTM Language Models», *CoRR*, vol. abs/1708.02182, 2017. arXiv: 1708.02182. [Online]. Available: http://arxiv.org/abs/1708.02182.

[75] S. Merity, C. Xiong, J. Bradbury, and R. Socher, «Pointer Sentinel Mixture Models», in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. [Online]. Available: https://openreview.net/forum?id=Byj72udxe.

[76] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space*, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781.

[77] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, «Recurrent neural network based language model», vol. 2, 2010, pp. 1045–1048.

[78] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, «Distributed Representations of Words and Phrases and their Compositionality», in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[79] T. Miyato, S. ichi Maeda, M. Koyama, and S. Ishii, «Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, pp. 1979–1993, 2019.

[80] T. Miyato, S.-I. Maeda, M. Koyama, and S. Ishii, «Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning», 2017. arXiv: 1704.03976 [stat.ML].

[81] R. Nallapati, F. Zhai, and B. Zhou, «SummaRuNNer: A Recurrent Neural Network Based Sequence Model for Extractive Summarization of Documents», in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17, San Francisco, California, USA, 2017, 3075–3081.

[82] M. A. Nielsen, *Neural Networks and Deep Learning*, misc, 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/.

[83] H. Ouchi, H. Shindo, and Y. Matsumoto, «A Span Selection Model for Semantic Role Labeling», in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018, pp. 1630–1642. DOI: 10.18653/v1/D18-1191. [Online]. Available: https://www.aclweb.org/anthology/D18-1191.

[84] S. J. Pan and Q. Yang, «A Survey on Transfer Learning», *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1345–1359, 2010.

[85] R. Pascanu, T. Mikolov, and Y. Bengio, «Understanding the exploding gradient problem», *CoRR*, vol. abs/1211.5063, 2012. arXiv: 1211.5063. [Online]. Available: http://arxiv.org/abs/1211.5063.

[86] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, «PyTorch: An Imperative Style, High-Performance Deep Learning Library», in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[87] R. Paulus, C. Xiong, and R. Socher, «A Deep Reinforced Model for Abstractive Summarization», *CoRR*, vol. abs/1705.04304, 2017. arXiv: 1705.04304. [Online]. Available: http://arxiv.org/abs/1705.04304.

[88] J. Pennington, R. Socher, and C. Manning, «Glove: Global Vectors for Word Representation», in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. [Online]. Available: https://www.aclweb.org/anthology/D14-1162.

[89] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, «Deep Contextualized Word Representations», in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2018, pp. 2227–2237. [Online]. Available: https://www.aclweb.org/anthology/N18-1202.

[90] J. Phang, T. Févry, and S. R. Bowman, «Sentence Encoders on STILTs: Supplementary Training on Intermediate Labeled-data Tasks», *ArXiv*, vol. abs/1811.01088, 2018.

[91] B. Plank and G. van Noord, «Effective Measures of Domain Similarity for Parsing», in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011, pp. 1566–1576. [Online]. Available: https://www.aclweb.org/anthology/P11-1157.

[92] O. Press and L. Wolf, «Using the Output Embedding to Improve Language Models», in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, 2017, pp. 157–163. [Online]. Available: https://www.aclweb.org/anthology/E17-2025.

[93] A. Radford, R. Józefowicz, and I. Sutskever, «Learning to Generate Reviews and Discovering Sentiment», *CoRR*, vol. abs/1704.01444, 2017. arXiv: 1704.01444. [Online]. Available: http://arxiv.org/abs/1704.01444.

[94] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, «Improving language understanding by generative pre-training», 2018. [Online]. Available: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

[95] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, «Language models are unsupervised multitask learners», *OpenAI Blog*, vol. 1, no. 8, 2019.

[96] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, «Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer», *arXiv e-prints*, 2019. arXiv: 1910.10683.

[97] S. Ruder, «Neural Transfer Learning for Natural Language Processing», PhD thesis, National University of Ireland, Galway, 2019.

[98] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, «Learning Representations by Back-propagating Errors», *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: 10.1038/323533a0. [Online]. Available: http://www.nature.com/articles/323533a0.

[99] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, «ImageNet Large Scale Visual Recognition Challenge», *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[100] D. Saxton, E. Grefenstette, F. Hill, and P. Kohli, «Analysing Mathematical Reasoning Abilities of Neural Models», *ArXiv*, vol. abs/1904.01557, 2019.

[101] M. Schuster and K. Nakajima, «Japanese and Korean voice search», in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 5149–5152.

[102] M. Schuster and K. Nakajima, «Japanese and Korean voice search», *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5149–5152, 2012.

[103] M. Schuster and K. Paliwal, «Bidirectional recurrent neural networks», *Signal Processing, IEEE Transactions on*, vol. 45, pp. 2673 –2681, 1997. DOI: 10.1109/78.650093.

[104] R. Sennrich, B. Haddow, and A. Birch, «Neural Machine Translation of Rare Words with Subword Units», in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. [Online]. Available: https://www.aclweb.org/anthology/P16-1162.

[105] M. J. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, «Bidirectional Attention Flow for Machine Comprehension», *CoRR*, vol. abs/1611.01603, 2016. arXiv: 1611.01603. [Online]. Available: http://arxiv.org/abs/1611.01603.

[106] I. V. Serban, T. Klinger, G. Tesauro, K. Talamadupula, B. Zhou, Y. Bengio, and A. Courville, «Multiresolution Recurrent Neural Networks: An Application to Dialogue Response Generation», in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17, San Francisco, California, USA, 2017, 3288–3294.

[107] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, «Mastering the Game of Go with Deep Neural Networks and Tree Search», *Nature*, vol. 529, no. 7587, pp. 484–489, 2016, ISSN: 0028-0836. DOI: 10.1038/nature16961.

[108] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, «Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank», in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2013, pp. 1631–1642. [Online]. Available: http://www.aclweb.org/anthology/D13-1170.

[109] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting», *J. Mach. Learn. Res.*, vol. 15, no. 1, 1929–1958, 2014, ISSN: 1532-4435.

[110] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, «End-To-End Memory Networks», 2015. arXiv: 1503.08895 [cs.NE].

[111] B. Sun, J. Feng, and K. Saenko, «Return of Frustratingly Easy Domain Adaptation», in *Proceedings of the Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, 2016, pp. 2058–2065. [Online]. Available: http://dl.acm.org/citation.cfm?id=3016100.3016186.

[112] C. Sun, X. Qiu, Y. Xu, and X. Huang, «How to Fine-Tune BERT for Text Classification?», *ArXiv*, vol. abs/1905.05583, 2019.

[113] I. Sutskever, O. Vinyals, and Q. V. Le, «Sequence to Sequence Learning with Neural Networks», in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., 2014, pp. 3104–3112. [Online]. Available: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

[114] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, «Intriguing properties of neural networks», in *International Conference on Learning Representations*, 2014. [Online]. Available: http://arxiv.org/abs/1312.6199.

[115] W. L. Taylor, «"Cloze Procedure": A New Tool for Measuring Readability», *Journalism Quarterly*, vol. 30, no. 4, pp. 415–433, 1953. DOI: 10.1177/107769905303000401. eprint: https://doi.org/10.1177/107769905303000401. [Online]. Available: https://doi.org/10.1177/107769905303000401.

[116] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, «Deep Domain Confusion: Maximizing for Domain Invariance», *CoRR*, vol. abs/1412.3474, 2014. arXiv: 1412.3474. [Online]. Available: http://arxiv.org/abs/1412.3474.

[117] V. Uení, T. V. Brně, G. A. Multimédií, and D. Práce, «Statistical Language Models Based on Neural Networks», 2012.

[118] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, «Attention Is All You Need», 2017. arXiv: 1706.03762 [cs.CL].

[119] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, «SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems», *CoRR*, vol. abs/1905.00537, 2019. arXiv: 1905.00537. [Online]. Available: http://arxiv.org/abs/1905.00537.

[120] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, «GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding», in *Proceedings of the Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for Natural Language Processing*, 2018, pp. 353–355. [Online]. Available: https://www.aclweb.org/anthology/W18-5446.

[121] H. Wang, Z. Gan, X. Liu, J. Liu, J. Gao, and H. Wang, «Adversarial Domain Adaptation for Machine Reading Comprehension», in *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2019, pp. 2510–2520. [Online]. Available: https://www.aclweb.org/anthology/D19-1254.

[122] A. Warstadt, A. Singh, and S. Bowman, «Neural Network Acceptability Judgments», *Transactions of the Association for Computational Linguistics*, vol. 7, no. 0, pp. 625–641, 2019. [Online]. Available: https://transacl.org/ojs/index.php/tacl/article/view/1710.

[123] L. Weng, «Attention? Attention!», *lilianweng.github.io/lil-log*, 2018. [Online]. Available: http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html.

[124] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, *HuggingFace's Transformers: State-of-the-art Natural Language Processing*, 2019. arXiv: 1910.03771 [cs.CL].

[125] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, «Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation», *CoRR*, vol. abs/1609.08144, 2016. arXiv: 1609.08144. [Online]. Available: http://arxiv.org/abs/1609.08144.

[126] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, «XLNet: Generalized Autoregressive Pretraining for Language Understanding», in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, Eds., 2019, pp. 5753–5763. [Online]. Available: http://papers.nips.cc/paper/8812-xlnet-generalized-autoregressive-pretraining-for-language-understanding.pdf.

[127] Z. Yang, Z. Hu, C. Dyer, E. P. Xing, and T. Berg-Kirkpatrick, «Unsupervised Text Style Transfer using Language Models as Discriminators», in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 7287–7298. [Online]. Available: http://papers.nips.cc/paper/7959-unsupervised-text-style-transfer-using-language-models-as-discriminators.pdf.

[128] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, «Hierarchical Attention Networks for Document Classification», San Diego, California, 2016, pp. 1480–1489.

[129] A. W. Yu, D. Dohan, Q. Le, T. Luong, R. Zhao, and K. Chen, «Fast and Accurate Reading Comprehension by Combining Self-Attention and Convolution», in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=B14TlG-RW.

[130] A. W. Yu, D. Dohan, M. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le, «QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension», *CoRR*, vol. abs/1804.09541, 2018. arXiv: 1804.09541. [Online]. Available: http://arxiv.org/abs/1804.09541.

[131] W. Zaremba, I. Sutskever, and O. Vinyals, *Recurrent Neural Network Regularization*, cite arxiv:1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329.

[132] K. Zhang and S. Bowman, «Language Modeling Teaches You More than Translation Does: Lessons Learned Through Auxiliary Syntactic Task Analysis», in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 2018, pp. 359–361. DOI: 10.18653/v1/W18-5448. [Online]. Available: https://www.aclweb.org/anthology/W18-5448.

[133] X. Zhang, J. Zhao, and Y. LeCun, «Character-level Convolutional Networks for Text Classification», in *Proceedings of the Conference on Neural Information Processing Systems*, 2015, pp. 649–657. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969239.2969312.