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

Towards Neural Models with System 2 - Type Capabilities

Diploma Thesis

of

PANAGIOTIS KOLIOS

Supervisor: Alexandros Potamianos
Associate Professor, NTUA

Athens, 05/2023

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

Towards Neural Models with System 2 - Type Capabilities

DIPLOMA THESIS

of

PANAGIOTIS KOLIOS

Supervisor: Alexandros Potamianos
Associate Professor, NTUA

Approved by the examination committee on 30/05/2023.

(Signature)	*(Signature)*	*(Signature)*
.....	.....	.....
Alexandros Potamianos	Konstantinos Tzafestas	Stefanos Kollias
Associate Professor, NTUA	Associate Professor, NTUA	Professor, NTUA

Athens, 05/2023

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

(Signature)

.....
Panagiotis Kolios

Electrical & Computer
Engineer

Περίληψη

Το θέμα αυτής της διπλωματικής εργασίας είναι η μελέτη και η ανάπτυξη ιδιοτήτων και χαρακτηριστικών των νευρωνικών δικτύων, οι οποίες προσδοκάται να τους επιτρέψουν να αποθηκεύουν και να διαχειρίζονται πληροφορία με τρόπο κοντινότερο σε αυτόν με τον οποίο το πράττει ο ανθρώπινος εγκέφαλος. Εστιάζουμε κυρίως στο γνωστό νευρωνικό μοντέλο transformer και σε παραλλαγές του, που έχουν χρησιμοποιηθεί κυρίως για την επίλυση προβλημάτων φυσικής γλώσσας (natural language).

Παρουσιάζουμε θέματα τα οποία άπτονται των περιοχών της γνωσιακής επιστήμης, της νευροεπιστήμης και της μελέτης της αιτότητας και σχετίζονται με τον τρόπο με τον οποίο σκέφτεται και λειτουργεί το ανθρώπινο μυαλό. Εξετάζουμε τον διαχωρισμό της ανθρώπινης νόησης σε System 1 και System 2, τον οποίο αναλύει ο Kahneman στο βιβλίο του Thinking Fast And Slow. Ακολούθως περιγράφουμε επιλεγμένα τμήματα του ανθρώπινου εγκεφάλου, καταδεικνύοντας την, υπαρκτή σε αυτόν, τάση εξειδίκευσης και καταμερισμού των εργασίων επεξεργασίας της πληροφορίας, και παραθέτουμε τα βασικά χαρακτηριστικά ενός θεωρητικού μοντέλου για τον τρόπο επικοινωνίας μεταξύ των διάφορων τμημάτων του, το οποίο ονομάζεται Global Workspace Theory. Επίσης, εξηγούμε πως συμπεράσματα από μελέτες στο πεδίο της αιτιότητας (causality) αιτιολογούν την οργάνωση της πληροφορίας σε ανεξάρτητους μηχανισμούς (independent mechanisms).

Στη συνέχεια μελετάμε προσπάθειες ενσωμάτωσης των παραπάνω ιδεών στα σύγχρονα νευρωνικά δίκτυα. Βασιζόμαστε στην δουλειά των Goyal και Bengio πάνω στις επαγωγικές προτιμήσεις (inductive biases), σκοπός των οποίων είναι ο καθορισμός των υποθέσεων που γίνονται κατά την διάρκεια της εκπαίδευσης ενός μοντέλου, καθώς και αυτών που κάνουν τα νευρωνικά δίκτυα όσον αφορά στην στατιστική και αιτιακή κατανομή των δεδομένων. Παρουσίαζουμε προσπάθειες ενσωμάτωσης διαφόρων τύπων επαγωγικών προτιμήσεων, είτε στην εκπαιδευτική διαδικασία είτε σε γνωστές αρχιτεκτονικές μοντέλων νευρωνικών δικτύων. Εστιάζουμε σε προσπάθειες που επιδιώκουν την εξειδίκευση τμημάτων των παραπάνω μοντέλων, κυρίως μέσω διαδικασιών ανταγωνισμού μεταξύ των τμημάτων αυτών.

Προτείνουμε δύο μετατροπές σε νευρωνικά δίκτυα που βασίζονται στο μοντέλο transformer. Αρχικά προτείνουμε την αντικατάσταση των δικτύων εμπρόσθιας τροφοδότητσης που βρίσκονται στα στρώματα του transformer από ένα σύνολο παράλληλων αντίστοιχων δικτύων, τα οποία θα εκπαιδεύονται μέσω ανταγωνιστικών διαδικασιών, οι νικητές των οποίων θα αποκτούν τις άδειες επεξεργασίας των αντίστοιχων στοιχείων του διανύσματος εισόδου. Ακόμα προτείνουμε την εφαρμογή ενός αντίστοιχου συστήματος για την εκπαίδευση των κεφαλών προσοχής των ίδιων μοντέλων.

Εφαρμόζουμε την δεύτερη μέθοδο, που αφορά στις κεφαλές προσοχής, στο μοντέλο transformer και το εκπαιδεύουμε πάνω σε πρόβλαματα μηχανικής μετάφρασης (neural machine translation), καθώς και στο μοντέλο BERT, το οποίο εκπαιδεύουμε στο πρόβλημα μοντελοποίησης της φυσικής γλώσσας (language modeling). Τα δύο μοντέλα δεν επιδεικνύουν σαφείς τάσεις βελτίωσης στα προβλήματα αυτά σε σχέση με τα μοντέλα βάσης (baseline models). Εξετάζουμε τα πιθανά αίτια αυτής της συμπεριφοράς και προτείνουμε πιθανές μεθόδους επίλυσης των προβλημάτων καθώς και κατευθύνσεις για μελλοντική έρευνα.

Λέξεις Κλειδιά

Βαθιά Μάθηση, Επεξεργασία Φυσικής Γλώσσας, Νευρωνικά Δίκτυα, Meta-Learning, Transform-

ers, BERT, Causality, Independent Mechanisms, Specialization, Modularity, Μετεκπαίδευση (Fine-Tuning)

Abstract

The subject of this thesis is the study and development of properties and characteristics of neural nets, that aim at allowing these models to store and manage information in a manner that resembles the way the human brain does so. Our study revolves around the well-known transformer model and its variants, which have mainly been applied to natural language problems.

We present topics coming from the areas of cognitive science, neuroscience and the study of causality and are related to the way the human mind thinks and works. We examine the division of the human mind into System 1 and System 2, which Kahneman analyzes in his book, Thinking Fast And Slow. We then describe selected parts of the human brain, demonstrating an existent tendency in it, towards specialization and division of information processing tasks, and list the main features of a theoretical model, whose goal is to describe the way the various brain parts communicate with each other, called Global Workspace Theory. We also explain how conclusions derived from studies in the field of causality justify the organization of information processing modules into independent mechanisms.

We subsequently examine the possibility of integrating the aforementioned ideas into modern neural networks. We build on the work of Goyal and Bengio on inductive biases, which aim to determine the assumptions made by the training algorithms regarding the training conditions, as well as the hypotheses made by the neural models regarding the (causal) model that has generated the data set. We present a set of attempts to incorporate various types of inductive preferences, either in the training process or in famous neural network model architectures. We focus on efforts that seek to promote the specialization of the various model areas, mainly through competitive processes between those areas, leading to a modular architectures.

We propose two modifications in the architectures of neural models that are based on the transformer model. We first propose the replacement of the feed-forward networks located in the transformer layers by a set of parallel networks of the same kind, which are trained through competitive processes, the winners of whose acquire the rights to process the respective elements of the input vector. We also propose the application of a similar system for the training of the attention heads of the same model, also based on a competition procedure among the heads of each layer.

We apply the second method, whose goal is to train specialized attention heads, to the transformer model and then train it on neural machine translation problems, as well as to the BERT model, which we train on a natural language modeling problem. The two models show no clear signs of improvement in these problems compared to the baseline models. We examine the possible causes of this behavior and suggest a variety of possible solutions to these problems as well as directions for future research.

Keywords

Deep Learning, Natural Language Processing, Neural Networks, Meta-Learning, Transformers, BERT, Causality, Independent Mechanisms, Specialization, Modularity, Fine-Tuning

To My Father,
Konstantinos

Ευχαριστίες

Ένα σύντομο σημείωμα για να εκφράσω τις ευχαριστίες μου στους ανθρώπους που με βοήθησαν για να ολοκληρώσω αυτήν την εργασία.

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή Αλέξανδρο Ποταμιάνο, ο οποίος με καθοδήγησε καθόλη την διάρκεια εκπόνησης της διπλωματικής μου εργασίας, δίνοντας μου παράλληλα την ευκαιρία ερευνήσω ένα μεγάλο φάσμα θεμάτων πριν εστιάσω στα θέματα τα οποία κέντρισαν τελικά την προσοχή μου.

Ακόμα θα ήθελα να ευχαριστήσω τους διδακτρικούς ερευνητές Γιώργο Παρασκευόπουλο και Ευθύμη Γεωργίου, οι οποίοι πρόθυμα συζητούσαν τις λεπτομέρειες της υλοποίησης της πρότασής μου, βοηθώντας με να διασφαλίσω την ορθότητά της.

Τέλος, οφείλω ένα μεγάλο ευχαριστώ στην μητέρα μου, Κατερίνα, και την αδελφή μου, Ήβη, οι οποίες μου στάθηκαν καθόλη την διάρκεια εκπόνησης της διπλωματικής μου και με παρότρυναν να συνεχίσω ανεξαρτήτως των όποιων δυσκολιών μπορεί να συναντούσα.

Παναγιώτης Κολιός

Αθήνα, Απρίλιος 2023

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 0

Εκτεταμένη Ελληνική Περίληψη

0.1 Εισαγωγή

0.1.1 Κίνητρα

Μελέτες στον τομέα της σύγχρονης νευροεπιστήμης έχουν δείξει πως η αποθήκευση των μεμονωμένων πληροφοριών δεν πραγματοποιείται από νευρώνες ομοιόμορφα κατανεμημένους στον ανθρώπινο εγκέφαλο, αλλά πως οι διάφορες περιοχές του εγκεφάλου παρουσιάζουν τάσεις εξειδίκευσης. Κάποια τμήματα του εγκεφάλου έχουν βρεθεί να συνδέονται εντονότερα με την διαχείριση πληροφορίας που σχετίζεται με συγκεκριμένες αισθήσεις, π.χ. ακοή ή αφή, και με την διαφοροποίηση να συνεχίζεται στον τρόπο διαχείρισης, π.χ. αποθήκευση ή επεξεργασία, καθώς και στον τύπο της πληροφορίας, π.χ. αντίδραση σε εξωτερικό ερέθισμα ή μετατροπή θέλησης σε πράξη. Επίσης, κατά την Global Workspace Theory [44, 45, 46, 47, 48], υπάρχουν ειδικευμένες ομάδες νευρώνων στον ανθρώπινο εγκέφαλο που δρουν σχετικά ανεξάρτητα η μία από την άλλη, και επικοινωνούν σποραδικά μέσω ενός ενιαίου καναλιού, στο οποίο εμπλέκεται και η συνείδηση.

Παρόλα αυτά, στα σημερινά νευρωνικά δίκτυα δεν υπάρχει μία μέθοδος η οποία να καθορίζει τον τόπο και τρόπο αποθήκευσης της πληροφορίας που αποκτάται κατά την εκπαίδευση, καθώς και την διαδικασία επιλογής των γνώσεων, οι οποίες κρίνεται ότι είναι οι πιο χρήσιμες την εκάστοτε χρονική στιγμή. Αντίθετα, οι παραπάνω λειτουργίες αναπτύσσονται αυτόματα κατά την διάρκεια της εκπαίδευσης με τρόπο που δεν είναι ελέγξιμος από τον άνθρωπο. Εκτός από το πρόβλημα της κακής χρήσης των διαθέσιμων πόρων, η συγκεκριμένη παράλειψη είναι πολύ πιθανό σχετίζεται με την πτώση της απόδοσης των νευρωνικών δικτύων όταν αυτά καλούνται να βγάλουν συμπεράσματα για δείγματα που ανήκουν στην κατανομή των παραδειγμάτων εκπαίδευσης αλλά όχι σε αυτά (in-distribution generalization), και με την αδυναμία τους να γενικεύσουν σε δείγματα άλλων κατανομών, οι οποίες έχουν κάποια σχέση με την πρώτη (out-of-distribution generalization).

Σημερινές θεωρίες που προέρχονται από τον τομέα της γνωσιακής επιστήμης (cognitive science), οι άνθρωποι γενικεύουν ανακτώντας κομμάτια γνώσης από τη μνήμη τους, τα οποία σχετίζονται με το παρόν πρόβλημα, και συνδυάζοντάς τα με καινούριους τρόπους. Είναι πολύ δύσκολο να φανταστεί κάποιος πως τα νευρωνικά δίκτυα θα μπορούσαν να αποκτήσουν τέτοια ευελιξία στην διαχείριση των μεμονωμένων τμημάτων πληροφορίας χωρίς να οργανωθούν κατάλληλα οι τρόποι αποθήκευσης, ανάκτησης και επεξεργασίας τους από μέρους των μοντέλων αυτών. Είναι λοιπόν λογικό να εξερευνήσει κάποιος μεθόδους που θα ωθούν τα νευρωνικά δίκτυα να διαχερίζονται τη γνώση κατά τρόπο παρόμοιο με αυτόν του ανθρωπίνου εγκεφάλου.

0.1.2 Συνεισφορές

Πραγματοποιούμε μία βιβλιογραφική ανασκόπηση της έρευνας που έχει πραγματοποιηθεί στην παραπάνω κατεύθυνση, επικεντρώνοντας την προσοχή μας κυρίως στην ομάδα του ερευνητή Yoshua Bengio, από το πανεπιστήμιο του Μόντρεαλ. Επίσης, γίνεται ανάλυση των πηγών έμπνευσής τους,

οι οποίες προέρχονται από τους κλάδους του συμπεριφορισμού, της γνωσιακής επιστήμης (cognitive science), της νευροεπιστήμης και της μελέτης αιτιακών μεταβλητών και μηχανισμών (causal variables and mechanisms).

Με βάση τα παραπάνω προτείνουμε δύο αρχιτεκτονικές μετατροπές σε νευρωνικά δίκτυα που βασίζονται στο μοντέλο μηχανικής μάθησης transformer [1] και εφαρμόζουμε την μία από τις δύο. Συγκεκριμένα, εγκαθιστούμε ένα σύστημα επιλογής στις κεφαλές (attention heads) των μηχανισμών προσοχής τους (attention mechanisms), το οποίο στοχεύει στην εξειδίκευσή τους μέσω ανταγωνιστικών διαδικασιών. Παράλληλα, προτείνουμε την εγκατάσταση ενός παρόμοιου συστήματος στα δίκτυα πρόσθιας τροφοδότησης (feed-forward neural networks) των μοντέλων αυτών, τα οποία συνήθως είναι εγκατεστημένα μετά τους μηχανισμούς προσοχής.

0.2 Το Μοντέλο Transformer

Κεντρικό σε αυτή την εργασία είναι το μοντέλο **transformer** (Vaswani et al. (2017) [1]), το οποίο χρησιμοποιήθηκε αρχικά για το πρόβλημα της μετάφρασης φυσικής γλώσσας, και έχει αποτελέσει τη βάση για τα σημερινά κορυφαία (state-of-the-art) μοντέλα που χρησιμοποιούνται για την επίλυση μίας μεγάλης γκάμας προβλημάτων φυσικής γλώσσας (natural language). Ο transformer είναι ένα νευρωνικό δίκτυο το οποίο διαχειρίζεται ακολουθίες στοιχείων τα οποία μπορεί να συνδέονται σημασιολογικά μεταξύ τους, όπως οι λέξεις σε ένα κείμενο. Το σύνολο όλων των πιθανών στοιχείων ονομάζεται λεξιλόγιο (dictionary) του προβλήματος και κάθε στοιχείο αναπαρίσταται από ένα διάνυσμα πραγματικών αριθμών (embedding).

Το μοντέλο, το οποίο φαίνεται στην εικόνα 1, αποτελείται από 2 τμήματα, έναν κωδικοποιητή (encoder) και έναν αποκωδικοποιητή (decoder). Ο κωδικοποιητής διαβάζει μία ακολουθία στοιχείων και δημιουργεί αναπαραστάσεις για αυτά με βάση τα συμφραζόμενα (contextual representations). Ο αποκωδικοποιητής διαβάζει αυτές τις αναπαραστάσεις και παράγει μία ακολουθία εξόδου. Για παράδειγμα, ένα ζεύγος ακολουθίας εισόδου-εξόδου μπορεί να είναι μία πρόταση και η μετάφρασή της σε μία άλλη γλωσσα.

Ο κωδικοποιητής αποτελείται από N επίπεδα με κοινή δομή, αποτελούμενα, το καθένα, από έναν μηχανισμό προσοχής (Bahdanau et al. (2014) [10]) ακολουθούμενο από ένα δίκτυο πρόσθιας τροφοδότησης. Η επεξεργασία κάθε στοιχείου πραγματοποιείται παράλληλα με αυτή των υπολοίπων. Επίσης η επεξεργασία όλων των στοιχείων από ένα επίπεδο πραγματοποιείται από νευρωνικές δομές με κοινές παραμέτρους, ανεξάρτητες της θέσεως του στοιχείου. Άρα ο μηχανισμός προσοχής ενός επιπέδου είναι κοινός για όλες τις θέσεις, όπως και το δίκτυο πρόσθιας τροφοδότησης, αλλά δύο μηχανισμοί που χρησιμοποιούνται σε διαφορετικά επίπεδα έχουν διαφορετικές παραμέτρους.

Οι μηχανισμοί προσοχής είναι υπεύθυνοι για την ανταλλαγή πληροφοριών μεταξύ των στοιχείων. Δέχονται ως εισόδους τις αναπαραστάσεις των στοιχείων $\mathbf{h}_i$, $i \in \{1, \cdots, T\}$, όπως αυτές προκύπτουν από το προηγούμενο επίπεδο $l-1$, και πολλαπλασιάζοντας τες με πί-

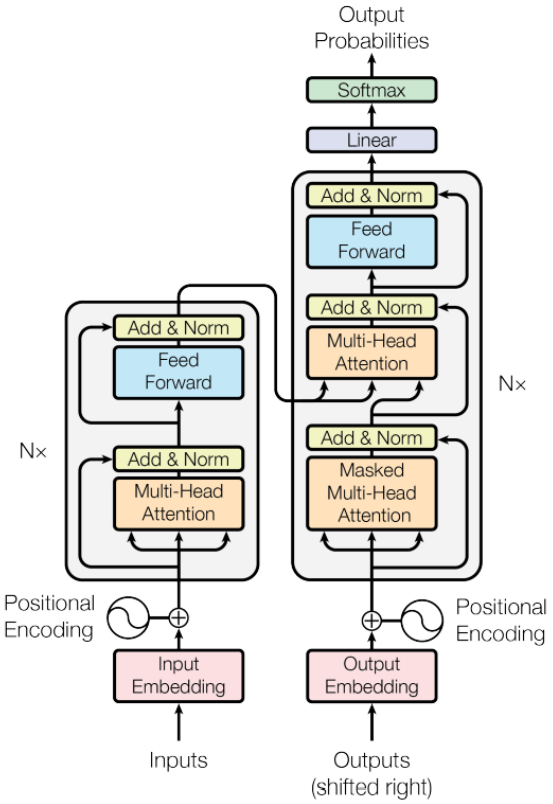


Figure 1. *Δομή του μοντέλου Transformer [1].*

νακες $\mathbf{W}_l^Q$, $\mathbf{W}_l^K$, $\mathbf{W}_l^V$ δημιουργούν διανύσματα $\mathbf{q}_i$, $\mathbf{k}_i$, $\mathbf{v}_i$ αντίστοιχα. Μετά υπολογίζουν τις τιμές συμβατότητας c_{ij} μεταξύ του στοιχείου iκαι κάθε στοιχείου j, συμπεριλαμβανομένου του i, για κάθε πιθανό i, περνώντας τα εσωτερικά γινόμενα $\mathbf{q}_i \cdot \mathbf{k}_j$, για κάθε j, από μία συνάρτηση softmax, $\text{softmax}(\mathbf{x}_i) = \frac{e^{\mathbf{x}_i}}{\sum_j e^{\mathbf{x}_j}}$. Για την δημιουργία της τελικής αναπαράστασης για το i υπολογίζεται το σταθμισμένο άθροισμα (weighted sum) των τιμών των στοιχείων $\mathbf{v}_j$ με τις αντίστοιχες τιμές συμβατότητας c_{ij}: $\tilde{\mathbf{h}}_i = \sum_j c_{ij}\mathbf{v}_j$. Στην πραγματικότητα ο παραπάνω μηχανισμός είναι μία έκδοση του μηχανισμού προσοχής, στην οποία η ακολουθία από την οποία προέρχονται τα διανύσματα $\mathbf{q}$ είναι ίδια με αυτήν από την οποία προέρχονται τα $\mathbf{k}$ και $\mathbf{v}$. Αυτού του τύπου ο μηχανισμός προσοχής ονομάζεται αυτο-προσοχή (self-attention).

Οι Vaswani et al. (2017) [1], για να επιτρέψουν σε μία λέξη μεγαλύτερη ευελιξία στην απόδοση τιμών συμβατότητας, πραγματοποιούν την παραπάνω διαδικασία πολλαπλές φορές χρησιμοποιώντας διαφορετικές παραμέτρους κάθε φορά: $\mathbf{W}_l^{Q,h}$, $\mathbf{W}_l^{K,h}$, $\mathbf{W}_l^{V,h}$, $h \in \{1, \cdots, H\}$. Οι αντίστοιχες έξοδοι $\tilde{\mathbf{h}}_i^h$ συνενώνονται για κάθε στοιχείο i, $\tilde{\mathbf{h}}_i = [\tilde{\mathbf{h}}_i^1, \tilde{\mathbf{h}}_i^2, \cdots, \tilde{\mathbf{h}}_i^H]$, και πολλαπλασιάζονται με ένα νέο πίνακα $\mathbf{W}_l^O$ για την δημιουργία της εξόδου του μηχανισμού προσοχής $\tilde{\mathbf{h}}_i^O = \mathbf{W}_l^O\tilde{\mathbf{h}}_i$. Τα επί μέρους τμήματα αυτού του μηχανισμού ονομάζονται κεφαλές (heads). Η αναπαράσταση κάθε στοιχείου, όπως προκύπτει από τον μηχανισμού προσοχής, περνάει μέσα από ένα δίκτυο πρόσθιας τροφοδότησης με ένα κρυφό επίπεδο: $\hat{\mathbf{h}}_i = \mathbf{W}_l^2 f(\mathbf{W}_l^1\tilde{\mathbf{h}}_i^O)$, όπου f μία συνάρτηση ενεργοποίησης (activation function).

Το μοντέλο επίσης χρησιμοποιεί μία τεχνική κανονικοποίησης (layer normalization [49]) των εσωτερικών αναπαραστάσεων που διατηρεί, καθώς και συνδέσεις παράκαμψης των νευρωνικών του δομών (residual connections [6]). Οι παραπάνω μέθοδοι συμβάλλουν στην καλύτερη εκπαίδευση του δικτύου, επιτρέποντας στον ερευνητή να στιβάξει περισσότερα νευρωνικά επίπεδα δημιουργώντας βαθιά νευρωνικά δίκτυα (deep networks).

Στα διανύσματα αναπαράστασης των λέξεων προστίθενται διανύσματα θέσης που κωδικοποιούν την θέση της κάθε λέξης μέσα στην πρόταση.

Ο αποκωδικοποιητής έχει ίδιο αριθμό επιπέδων και παρόμοια δομή με τον κωδικοποιητή με την διαφορά ότι ανάμεσα στους δύο προαναφερθέντες μηχανισμούς κάθε επιπέδου υπάρχει ένας ακόμα μηχανισμός προσοχής, στον οποίο τα διανύσματα $\mathbf{q}$ προέρχονται από τις εξόδους του προηγούμενου μηχανισμού αυτό-προσοχής του αποκωδικοποιητή και τα $\mathbf{k}$ και $\mathbf{v}$ προέρχονται από τις εξόδους του κωδικοποιητή. Ένα χαρακτηριστικό του αποκωδικοποιητή είναι ότι για η παραγωγή ακολουθιών πραγματοποιείται σειριακά, στοιχείο ανά στοιχείο. Για την παραγωγή κάθε νέου στοιχείου ο αποκωδικοποιητής συμβουλεύεται τα προηγούμενα στοιχεία που έχει παράξει, χρησιμοποιώντας τα ως εισόδους. Άρα ο παραπάνω μηχανισμός συνδέει τα στοιχεία που έχουν παραχθεί μέχρι μία στιγμή από τον αποκωδικοποιητή με τις αναπαραστάσεις των στοιχείων της ακολουθίας εισόδου. Ο μηχανισμός προσοχής αυτός ονομάζεται μηχανισμός προσοχής κωδικοποιητή-αποκωδικοποιητή.

Κατά τα άλλα ο κωδικοποιητής και ο αποκωδικοποιητής είναι παρόμοιοι ως προς τα χαρακτηριστικά των αρχιτεκτονικών τους.

0.2.1 Το Μοντέλο BERT

Οι Devlin et al. (2018) [11] εκπαιδεύουν έναν κωδικοποιητή, παρόμοιο με αυτόν στον transformer, στα προβλήματα της μοντελοποίησης γλώσσας (language modeling), δηλαδή στην πρόβλεψη κάποιων κρυμμένων λέξεων, και πρόβλεψης της επόμενης πρότασης (next sentence prediction). Ο εκπαιδευμένος κωδικοποιητής δημιουργεί αυτό που ονομάστηκε bidirectional encoder representations from transformers (**BERT**). Η ιδέα είναι ότι μέσω της διαδικασίας εκπαίδευσης έχει αποκτήσει γνώση της φυσικής γλώσσας και μπορεί να εκπαιδευτεί περαιτέρω και γρήγορα σε δεδομένα άλλων προβλημάτων φυσικής γλώσσας, π.χ. αναγνώριση συναισθήματος, ερωτο-απαντήσεις και περίληψη, τα οποία μπορεί να μην είναι αρκετά σε πλήθος για να εκπαιδεύσουν ένα μοντέλο εκ του μηδενός.

0.3 Σχετική Βιβλιογραφία

Ο Kahneman στο βιβλίο του Thinking Fast And Slow [50] διαχωρίζει νοητικά την διαδικασία της ανθρώπινης σκέψης σε δύο υποσυστήματα με σκοπό να εξηγήσει συγκεκριμένες ανθρώπινες συμπεριφορές. Το πρώτο υποσύστημα το ονομάζει Σύστημα 1 (System 1) και περιγράφει την λειτουργία του ως αυτόματη, γρήγορη και υποσυνείδητη. Το δεύτερο το ονομάζει Σύστημα 2 (System 2) και του αναθέτει λειτουργικότητες που πραγματοποιούνται συνειδητά και απαιτούν προσοχή και προσπάθεια.

Ο Bengio και η ομάδα του θεωρούν ότι τα σημερινά νευρωνικά δίκτυα εκπαιδεύονται πετυχημένα στις λειτουργικότητες του Συστήματος 1 αλλά όχι του Συστήματος 2, τις οποίες συσχετίζουν με την προαναφερθείσα ανθρώπινη ικανότητα γενίκευσης. Προτείνουν λοιπόν μία λίστα επιλογών, τις οποίες ονομάζουν επαγωγικές προτιμήσεις (inductive biases [51]) και των οποίων η πετυχημένη υλοποίηση και ενσωμάτωση στα υπάρχοντα νευρωνικά δίκτυα θεωρούν ότι θα ξεκλειδώσει τις ικανότητες που επιτυγχάνονται με τη χρήση του Συστήματος 2.

Η λίστα αυτή υιοθετεί αρκετές ιδέες από την επιστήμη της αιτιότητας (causality), που μελετά τις σχέσεις μεταξύ αιτιακών μεταβλητών, οι οποίες παίζουν τους ρόλους αιτιών και αποτελεσμάτων. Κατά τους Peters et al. (2018) [19], οι αιτιακές μεταβλητές συνδέονται με αιτιακούς μηχανισμούς, οι οποίοι περιγράφουν υποσυστήματα του πραγματικού κόσμου και τους αποδίδουν το ιδιαίτερο χαρακτηριστικό της ανεξαρτησίας μεταξύ τους. Το τελευταίο σημαίνει ότι γνώση για έναν είναι άχρηστη για την περιγραφή οποιουδήποτε άλλου και η αλλαγή ενός δεν σχετίζεται με οποιαδήποτε αλλαγή πραγματοποιείται σε κάποιον άλλο.

Οι Bengio και Goyal (2022) [51] θεωρούν ότι το ανθρώπινο μυαλό εκμεταλλεύεται την ύπαρξη αυτών των μηχανισμών στον πραγματικό κόσμο και τους αποθηκεύει στην μνήμη ως μοντέλα του κόσμου με τρόπο τέτοιον, που να του επιτρέπει να τους ανακαλεί όποτε χρειάζεται και να τους συνδυάζει για την επίλυση νέων προβλημάτων. Οι Bengio et al. (2020) [25] υποθέτουν ότι κάθε αλλαγή στο περιβάλλον, η οποία φέρνει τους ανθρώπους αντιμέτωπους με ένα νέο πρόβλημα, προκύπτει από την αλλαγή ενός πολύ μικρού τμήματος των υποκείμενων μηχανισμών που διέπουν την λειτουργία του. Αυτό οφείλεται εν μέρει στην προαναφερθείσα υπόθεση ανεξαρτησίας τους. Έτσι, οι άνθρωποι χρειάζεται να αλλάξουν μόνο ένα μικρό τμήμα του μοντέλου του κόσμου το οποίο διατηρούν στο μυαλό τους για να προσαρμοστούν στην αλλαγή. Με αυτόν τον τρόπο μπορούν να γενικεύουν γρήγορα και να ανταπεξέρχονται αποτελεσματικά στις νέες συνθήκες. Οι Bengio et al. (2020) [25] εκμεταλλεύονται το γεγονός της γρήγορης εκπαίδευσης ενός μοντέλου, το οποίο περιγράφει ένα σύστημα του οποίου λίγες συνιστώσες έχουν αλλάξει, για να μάθουν την ορθή δομή του συστμήματος αυτού, δηλαδή τα αληθινά αίτια κάθε αιτιακής μεταβλητής. Για να το κάνουν αυτό χρησιμοποιούν την ταχύτητα προσαρμογής του μοντέλου στην αλλαγή του συστήματος ως σήμα εκπαίδευσης σε έναν αλγόριθμο μετα-μάθησης (meta-learning).

Η μετα-μάθηση είναι μία μέθοδος εκπαίδευσης κατά την οποία στο μοντέλο παρουσιάζονται πολλά διαφορετικά προβλήματα, το ένα μετά το άλλο. Καλούμενο να τα επιλύσει με αποδοτικό τρόπο το μοντέλο μαθαίνει έναν γενικό τρόπο διαχείρισης τέτοιου είδους προβλημάτων [21]. Οι Bengio και Goyal (2022) [51] προτείνουν την μετα-μάθηση ως ένα πιθανό πλαίσιο μάθησης για νευρωνικά δίκτυα που εκπαιδεύονται με σκοπό την απόκτηση λειτουργικοτήτων του Συστήματος 2. Έτσι κάνουμε μία συνοπτική παρουσίαση των μεθόδων μετα-μάθησης και δίνουμε παραδείγματα γνωστών εφαρμογών.

Οι Parascandolo et al. (2017) [20] χρησιμοποιούν ανταγωνισμό για να ανακαλύψουν τους υποκείμενους αιτιακούς μηχανισμούς. Η λογική πίσω από την χρήση ανταγωνισμού βασίζεται είναι ότι ο νικητής του διαγωνισμού, ο οποίος διαγωνισμός αφορά την άδεια για την διαχείριση ενός δείγματος, βελτιώνεται στο πρόβλημα διαχείρισης δειγμάτων του ίδιου τύπου και αποκτά προβάδισμα σε σχέση με τους ανταγωνιστές του. Έτσι, έχει περισσότερες πιθανότητες να κάνει την καλύτερη πρόταση για δείγματα της ίδιας κατηγορίας στο μέλλον και άρα να ξαναχρησιμοποιηθεί για αυτά. Οι Goyal et al. (2019) [2], εμπνεόμενοι από την παραπάνω δουλειά, χρησιμοποιούν ανταγωνισμό για να εκπαιδεύσουν

αναδρομικά δίκτυα, καθένα από τα οποία επιδιώκεται να μοντελοποιήσει μία διαδικασία. Για το λόγο αυτό τα ονομάζουν ανεξάρτητους αναδρομικούς μηχανισμούς (recurrent independent mechanisms) (εικόνα 2). Ένας μηχανισμός προσοχής χρησιμοποιείται στην είσοδο για να βρεθούν τα πιο σχετικά με αυτήν δίκτυα, τα οποία και τελικά είναι τα μόνα που εκπαιδεύονται, μαθαίνοντας να διαχειρίζονται καλύτερα εισόδους του αντίστοιχου είδους.

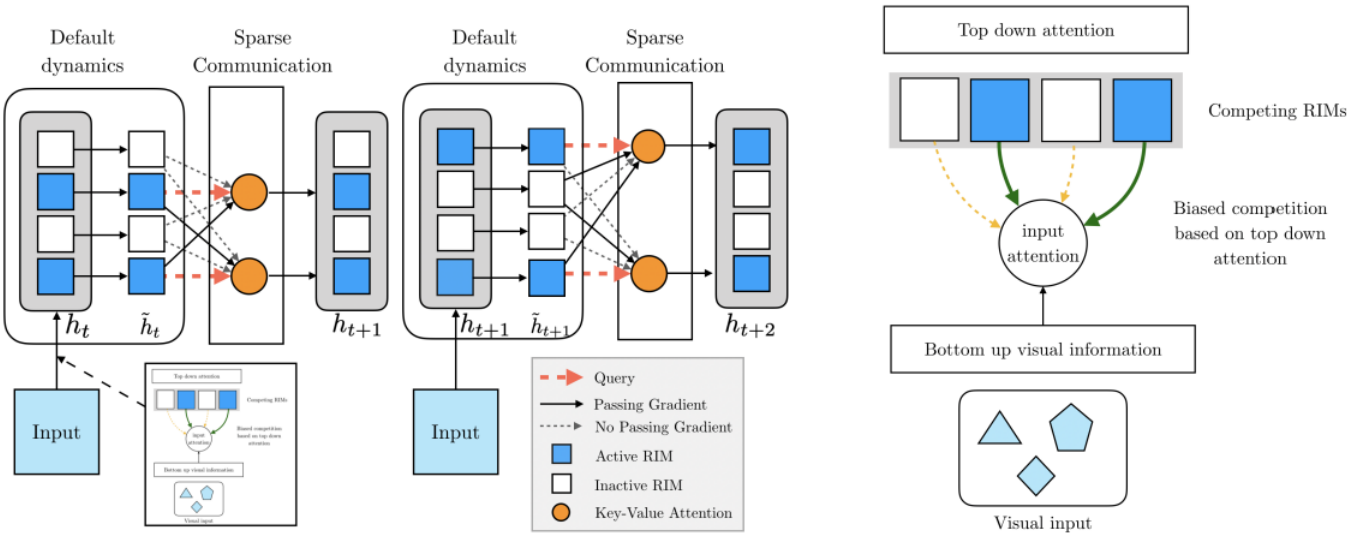


Figure 2. *Δομή ενός μοντέλου RIMs με 4 μηχανισμούς. Οι μηχανισμοί οι οποίοι προσέχουν περισσότερο τα δεδομένα ενεργοποιούνται (**δεξιά**). Οι ενεργοί μηχανισμοί, που απεικονίζονται με μπλε, εκτελούν ένα βήμα βάσει του εσωτερικού τους μοντέλου και αλληλεπιδρούν με τους υπόλοιπους (**αριστερά**). Οι ανενεργοί μηχανισμοί, που απεικονίζονται με λευκό, δεν ενημερώνουν την εσωτερική τους κατάσταση [2].*

Οι Rahaman et al. (2021) [27], βασιζόμενοι στο μοντέλο του transformer [1], προτείνουν τους neural interpreters, οι οποίοι εκπαιδεύουν δομικά στοιχεία, τα οποία ονομάζουν συναρτήσεις (functions). Οι συναρτήσεις κωδικοποιούν ξεχωριστές λειτουργικότητες. Μάλιστα, ορίζονται με τέτοιο τρόπο που επιτρέπει την προσθήκη καινούριων όποτε υπάρχει ανάγκη, καθώς και την αλλαγή του τρόπου που συνδυαστικά επεξεργάζονται τα στοιχεία της εισόδου με σκοπό την επίλυση καινούριων προβλημάτων.

Τέλος, οι Lamb et al. (2021) [28] εφαρμόζουν μία αρχιτεκτονική μετατροπή στο BERT (Devlin et al. (2018) [11]) για να δημιουργήσουν ανεξάρτητους μηχανισμούς που ανταγωνίζονται ο ένας τον άλλο για την δυνατότητα επεξεργασίας κάθε στοιχείου του διανύσματος εισόδου ξεχωριστά. Τα πειράματα που πραγματοποιούν δείχνουν βελτιώσεις τόσο στην ικανότητα των μοντέλων να γενικεύουν σε in-distribution όσο και σε out-of-distribution συνθήκες.

0.4 Προτεινόμενες Μετατροπές

Προτείνουμε δύο μετατροπές με σκοπό την βελτίωση της απόδοσης των νευρωνικών μοντέλων που βασίζονται στους transformers. Οι μετατροπές αυτές είναι στο πνεύμα των παραπάνω ερευνητικών προσπαθειών, αλλά αποτελούν μικρότερου μεγέθους αλλαγές στις δομές των μοντέλων συγκριτικά με τις προαναφερθείσες προσπάθειες.

0.4.1 Ανταγωνιζόμενα Δίκτυα Πρόσθιας Τροφοδότησης

Αρχικά, προτείνουμε την αντικατάσταση των δικτύων πρόσθιας τροφοδότησης στις αρχιτεκτονικές βασισμένες στο μοντέλο transformer από μία σειρά N_f παράλληλων τέτοιων δικτύων (εικόνα 3a). Για

να ωθήσουμε κάθε ένα από αυτά στην εξειδίκευση προτείνουμε την κατανομή της επεξεργασίας των στοιχείων εισόδου μεταξύ των παραπάνω δικτύων μέσω ενός μηχανισμού προσοχής. Ουσιαστικά ο μηχανισμός προσοχής υλοποιεί μία μορφή ανταγωνισμού (Parascandolo et al. (2017) [20]) μεταξύ των δικτύων πρόσθιας τροφοδότησης, με κάποια στοιχεία ομοιότητας με αυτήν των Lamb et al. (2021) [28].

Συγκεκριμένα, υιοθετώντας τις ονομασίες που χρησιμοποιήθηκαν από τους Rahaman et al. (2021) [27], σε κάθε δίκτυο πρόσθιας τροφοδότησης ενός επιπέδου l, F_{li}, $i \in \{1, \cdots, N_f\}$, ανατίθεται ένα διάνυσμα $\mathbf{f}_{li} \in \mathbb{R}^{D_f}$, το οποίο ονομάζουμε συνάρτηση. Ένα νέο δίκτυο πρόσθιας τροφοδότησης, InfF_l, χρησιμοποιεί την έξοδο του μηχανισμού προσοχής του ίδιου επιπέδου l σε κάθε θέση j για να παράξει ένα διάνυσμα $\mathbf{k}_{lj} = \text{InfF}_l(\mathbf{h}_{lj}) \in \mathbb{R}^{D_f}$. Τα διανύσματα $\{\mathbf{k}_{lj}\}_{j=1}^{T}$ και τα χαρακτηριστικά διανύσματα ανήκουν στον ίδιο διανυσματικό χώρο $\mathcal{S}$. Υπολογίζονται τιμές συμβατότητας μεταξύ κάθε στοιχείου εισόδου $\mathbf{h}_{lj}$, που εκπροσωπείται από το $\mathbf{k}_{lj}$, και των δικτύων πρόσθιας τροφοδότησης που εκπροσωπούνται από τα χαρακτηριστικά διανύσματα $\{\mathbf{f}_{li}\}_{i=1}^{T}$:

$$Cf_{lij} = \text{softmax}_i(\frac{\mathbf{f}_{li}\mathbf{k}_{lj}}{\sqrt{D_f}}) \tag{1}$$

όπου η softmax υπολογίζεται κατά μήκος του άξονα i που καταμετρά τα δίκτυα πρόσθιας τροφοδότησης.

Μετά την εφαρμογή των δικτύων αυτών στις αναπαραστάσεις της εισόδου $\mathbf{h}_{lj}$, η τελική έξοδος της δομής υπολογίζεται ως το βεβαρυμμένο άθροισμα των εξόδων τους:

$$\mathbf{o}_{lj} = \sum_{i=1}^{N_f} Cf_{lij} \cdot F_{li}(\mathbf{h}_{lj}) \tag{2}$$

0.4.2 Ανταγωνιζόμενες Κεφαλές Προσοχής

Όπως και στην περίπτωση των ανταγωνιζόμενων δικτύων πρόσθιας τροφοδότησης, είναι λογικό να προσπαθήσουμε να εγκαθιδρύσουμε μία μορφή ανταγωνισμού μεταξύ των κεφαλών προσοχής των μοντέλων που βρίσκονται στην αρχιτεκτονική του transformer (εικόνα 3b). Ερευνητικές προσπάθειες έχουν δείξει ότι οι κεφαλές υπο-χρησιμοποιούνται και πολλές κωδικοποιούν παρόμοια χαρακτηριστικά της φυσικής γλώσσας (Cordonnier et al. (2021) [13]) οδηγώντας σε πλεονασμό. Υποθέτουμε ότι δημιουργώντας ένα σύστημα αναγωνισμού μεταξύ των κεφαλών, όσον αφορά στο δικαίωμα επεξεργασίας της κάθε θέσης εισόδου, θα τις ωθήσουμε να εξειδικευτούν σε διαφορετικά χαρακτηριστικά της φυσικής γλώσσας.

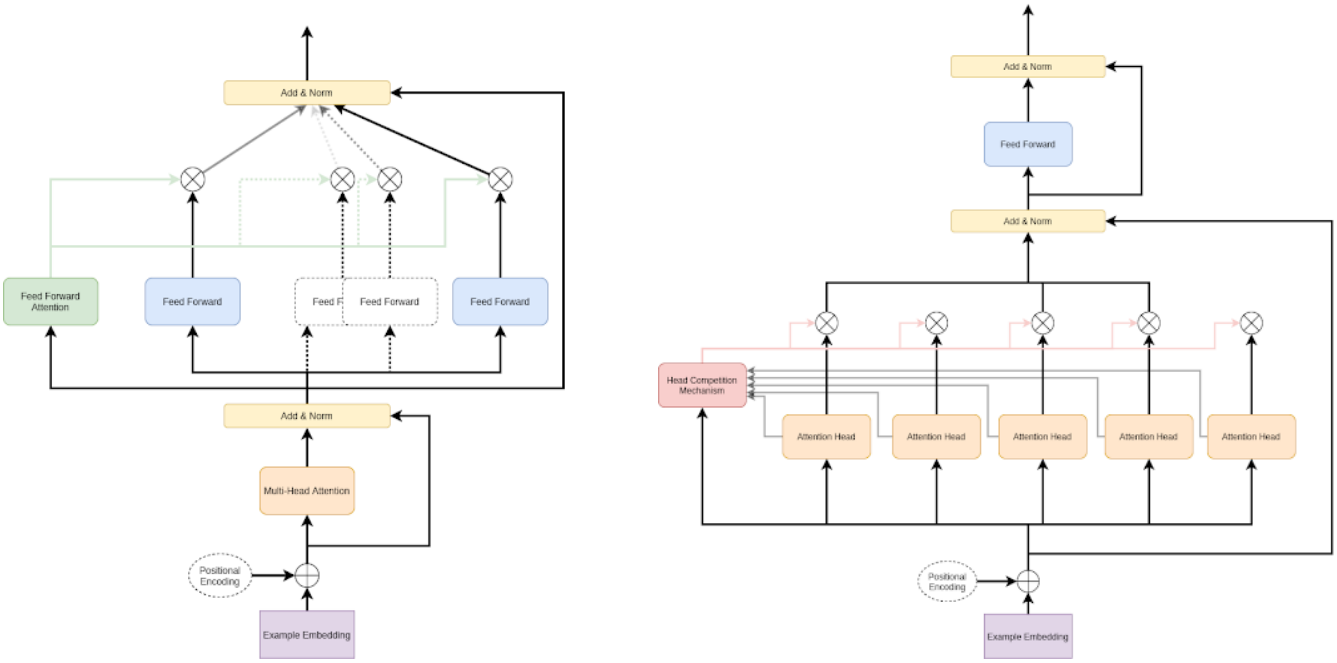
Πάλι, κάποιος μπορεί να χρησιμοποιήσει έναν μηχανισμό προσοχής για να εφαρμόσει τον προαναφερθέντα ανταγωνισμό. Για κάθε κεφαλή H_{li}, $i \in \{1, \cdots, H\}$, ένα χαρακτηριστικό διάνυσμα $\mathbf{a}_{li} \in \mathbb{R}^{D_a}$ αρχικοποιείται όπως και στην προηγούμενη εφαρμογή. Για την αναπαράσταση κάθε στοιχείου εισόδου $\mathbf{h}_{lj}$ χρησιμοποιείται ένα νέο δίκτυο πρόσθιας τροφοδότησης, InfA_l, το οποίο παράγει ένα διάνυσμα $\mathbf{k}_{lj} = \text{InfA}_l(\mathbf{h}_{lj}) \in \mathbb{R}^{D_a}$. Τα διανύσματα $\{\mathbf{k}_{lj}\}_{j=1}^{T}$ και τα χαρακτηριστικά διανύσματα ανήκουν στον ίδιο διανυσματικό χώρο $\mathcal{A}$. Υπολογίζονται τιμές συμβατότητας μεταξύ κάθε στοιχείου εισόδου $\mathbf{h}_{lj}$, που εκπροσωπείται από το $\mathbf{k}_{lj}$ και των κεφαλών προσοχής H_{li}, $i \in \{1, \cdots, H\}$, που εκπροσωπούνται από τα χαρακτηριστικά διανύσματα $\{\mathbf{a}_{li}\}_{i=1}^{T}$:

$$Ch_{lij} = \text{softmax}_i(\frac{\mathbf{a}_{li}\mathbf{k}_{lj}}{\sqrt{D_a}}) \tag{3}$$

όπου η softmax υπολογίζεται κατά μήκος του άξονα i που καταμετρά τις κεφαλές προσοχής.

Η έξοδος μία κεφαλής H_{li} για την θέση j δίνεται από το $\mathbf{o}_{lij} = Ch_{lij} \cdot \tilde{\mathbf{h}}_{lj}$, το οποίο είναι ουσιαστικά πολλαπλασιασμός αριθμού-διανύσματος και όπου $\tilde{\mathbf{h}}_{lj}$ θα ήταν η έξοδος της κεφαλής αν

δεν χρησιμοποιούνταν ο μηχανισμός ανταγωνιζόμενων κεφαλών προσοχής. Ονομάζουμε τις κεφαλές αυτές **ανταγωνιζόμενες κεφαλές προσοχής (ΑΚΠ)**.



(a) *Ανταγωνιζόμενα Δίκτυα Πρόσθιας Τροφοδότησης* **(b)** *Ανταγωνιζόμενες Κεφαλές Προσοχής*

Figure 3. *Οι δύο προτεινόμενες μετατροπές*

0.5 Πειράματα

Εφαρμόζουμε την δεύτερη πρόταση σε δύο μοντέλα, τον transformer και το RoBERTa [52], ένα μοντέλο βασισμένο στο BERT με βελτιστοποιημένες υπερ-παραμέτρους και διαδικασία εκπαίδευσης. Σε αυτή την εργασία εξετάζουμε την απόδοση των μοντέλων στην γενίκευση σε παραδείγματα που προέρχονται από την ίδια κατανομή με αυτά που χρησιμοποιούνται για την εκπαίδευση των μοντέλων.

Οι μέθοδοι υλοποιούνται και ενσωματώνονται στα μοντέλα με την βιβλιοθήκη FAIRSEQ [53].

0.5.1 Εφαρμογή στον Transformer

Εφαρμόζουμε την μέθοδο ΑΚΠ στις κεφαλές αυτό-προσοχής του κωδικοποιητή, του αποκωδικοποιητή και στις κεφαλές του μηχανισμού προσοχής κωδικοποιητή-αποκωδικοποιητή ενός μοντέλου transformer 6 επιπέδων, με διάσταση εσωτερικών σημάτων ίση με $D_{model} = 512$. Το μοντέλο εκπαιδεύεται στο πρόβλημα της μετάφρασης φυσικής γλώσσας και συγκεκριμένα στα δεδομένα του συνόλου εκπαίδευσης WSLT_14 από αγγλικά σε γερμανικά (en-de).

Μελετάμε την επίδραση του μηχανισμού στην απόδοση του μοντέλου για κάθε δυνατό συνδυασμό χρήσης ή όχι χρήσης του στους τρεις μηχανισμούς προσοχής. Επίσης, εξετάζουμε και τις επιδόσεις ενός μοντέλου με 8, αντί για 4 κεφαλές, θεωρώντας ότι, αν επιτυγχάνεται η εξειδίκευσή τους, τότε η αύξηση του αριθμού τους μπορεί να αποδειχθεί επωφελής. Τα αποτελέσματα παρατίθενται στον πίνακα 1.

Εκτός από μία μικρή βελτίωση των αποτελεσμάτων στις περιπτώσεις εφαρμογής της μεθόδου μόνο στις κεφαλές αυτό-προσοχής του κωδικοποιητή (33.61) και σε αυτές κωδικοποιητή και του αποκωδικοποιητή ταυτόχρονα (33.54), για ένα μοντέλο με 4 κεφαλές, δεν παρατηρείται δεν παρατηρείται άλλη σημαντική αύξηση της μετρικής BLEU στις υπόλοιπες περιπτώσεις.

Enc. SA	Dec. SA	Enc.-Dec. Attn	# of Heads	PPL Training	PPL Test	BLEU Test
			4	5.22	5.45	33.25
+			4	4.82	5.41	33.61
	+		4	5.14	5.46	33.28
+	+		4	4.66	5.42	33.54
		+	4	5.30	5.52	33.32
+		+	4	4.95	5.46	33.20
	+	+	4	5.13	5.44	33.46
+	+	+	4	4.82	5.43	33.47
			8	5.24	5.48	33.16
+			8	4.78	5.53	32.83
	+		8	5.49	5.63	32.41
+	+		8	4.64	5.55	32.89
		+	8	5.33	5.59	33.19
+		+	8	5.15	5.69	32.68
	+	+	8	5.14	5.52	33.18
+	+	+	8	4.48	5.51	33.28

Table 1. *Εφαρμογή της μεθόδου CAH στους τρεις μηχανισμούς προσοχής ενός μοντέλου transformer με* $N = 6$, $D_{model} = 512$, $D_{ffnn} = 1024$, $D_a = 24$, $D_{InfA} = 128$. *Enc. SA. σημαίνει μηχανισμός αυτο-προσοχής κωδικοποιητή, Dec. SA. μηχανισμός αυτο-προσοχής του αποκωδικοποιητή και Enc.-Dec. είναι ο μηχανισμούς προσοχής κωδικοποιητή-αποκωδικοποιητή. Ένας σταυρός + χρησιμοποιείται για να υποδηλώσει τη χρήση της μεθόδου CAH στον αντίστοιχο μηχανισμό και ένα κενό για να υποδηλώσει την απουσία της. Αναφέρεται η μετρική perplexity στο σύνολο δεδομένων εκπαίδευσης (PPL Training) και στο σύνολο δεδομένων ελέγχου (PPL Test) και η μέτρηση BLEU στο σύνολο δεδομένων ελέγχου. Το μοντέλο εκπαιδεύτηκε για 15 εποχές.*

Απενεργοποίηση Της Λιγότερο Σχετικής Κεφαλής

Εξετάζουμε ακόμα την περίπτωση ολικής απενεργοποίησης (εφαρμογή μηδενικού συντελεστή) της λιγότερο χρήσιμης κεφαλής και την εφαρμογή της συνάρτησης softmax στις υπόλοιπες. Πάλι δεν υπάρχουν ξεκάθαρα σημάδια βελτίωσης, όπως φαίνεται στον πίνακα 2.

Enc. SA	Dec. SA	Enc.-Dec. Attn	PPL Training	PPL Test	BLEU Test
			5.22	5.43	33.25
+			4.85	5.45	33.51
	+		5.16	5.45	33.41
+	+		4.73	5.49	33.23
		+	5.33	5.54	33.19
+		+	5.02	5.52	33.25
	+	+	5.19	5.51	33.33
+	+	+	4.83	5.45	33.42

Table 2. *Εφαρμογή της μεθόδου CAH στους τρεις μηχανισμούς προσοχής ενός μοντέλου transformer με* $N = 6$, $D_{model} = 512$, $D_{ffnn} = 1024$, $D_a = 24$, $D_{InfA} = 128$, $H = 4$. *Αφού το εσωτερικό γινόμενο μέσα στο softmax στην εξίσωση 5.3 υπολογιστεί για τα χαρακτηριστικά διανύσματα όλων των κεφαλών προσοχής και του κλειδιού που αντιπροσωπεύει ένα στοιχείο μιας συγκεκριμένης θέσης i, η κεφαλή στην οποία αντιστοιχεί το μικρότερο αντίστοιχο εσωτερικό γινόμενο απενεργοποιείται για τη θέση i και τα εσωτερικά γινόμενα για τις υπόλοιπες κεφαλές χρησιμοποιούνται για τον υπολογισμό τιμών συμβατότητας για αυτές. Αναφέρεται η μετρική perplexity στο σύνολο δεδομένων εκπαίδευσης (PPL Training) και στο σύνολο δεδομένων ελέγχου (PPL Test) και η μέτρηση BLEU στο σύνολο δεδομένων ελέγχου. Το μοντέλο εκπαιδεύτηκε για 15 εποχές.*

Εκπαίδευση ΑΚΠ Και Υπόλοιπου Μοντέλου Ξεχωριστά

Δοκιμάζουμε ακόμα να εκπαιδεύσουμε το μοντέλο και τον μηχανισμό υπολογισμού των τιμών συμβατότητας σε ξεχωριστά δεδομένα ώστε να προσπαθήσουμε να αποφύγουμε την προσαρμογή του μηχανισμού ΑΚΠ στις κεφαλές του μηχανισμού προσοχής του μοντέλου (overfitting). Διαχωρίζουμε το σύνολο δεδομένων σε ένα μεγάλο και ένα μικρό τμήμα (μία τάξη μεγέθους μικρότερο). Χρησιμοποιούμε δύο μεθόδους:

- Επαναληπτική εκπαίδευση (Iterative Training): Εκπαιδεύουμε για μία εποχή το μοντέλο χωρίς να αλλάζουμε τον μηχανισμό ΑΚΠ. Στην επόμενη εποχή εκπαιδεύουμε τον μηχανισμό ΑΚΠ μόνο και σε διαφορετικό υποσύνολο δεδομένων εκπαίδευσης κρατώντας τα βάρη του υπόλοιπου μοντέλου σταθερά. Επαναλαμβάνουμε τις δύο εποχές εκπαίδευσης χρησιμοποιώντας τα ίδια δύο υποσύνολα δεδομένων κάθε φορά πραγματοποιώντας 15 τέτοιες επαναλήψεις συνολικά.

- Σειριακή Εκπαίδευση (Sequential Training): Εκπαιδεύουμε για 15 εποχές το μοντέλο χωρίς να αλλάζουμε τον μηχανισμό ΑΚΠ, δηλαδή διατηρώντας τα βάρη του όπως στην αρχικοποίησή τους. Ακολούθως εκπαιδεύουμε τον μηχανισμό μόνο για 5 εποχές στο μικρό υποσύνολο δεδομένων εκπαίδευσης.

Εφαρμόζουμε τις μεθόδους εκπαίδευσης στις κεφαλές των δύο μηχανισμών αυτό-προσοχής του μοντέλου. Τα αποτελέσματα φαίνονται στον πίνακα 3. Τα αποτελέσματα είναι ελαφρώς χειρότερα από αυτά του μοντέλου, το οποίο εκπαιδεύεται ταυτόχρονα με τον μηχανισμό ΑΚΠ.

Enc. SA	Dec. SA	Training Mode	PPL Training	PPL Test	BLEU Test
+		Jointly	4.82	5.41	33.61
	+	Jointly	5.14	5.46	33.28
+	+	Jointly	4.66	5.42	33.54
+		Sequential	5.35	5.77	32.10
	+	Sequential	5.39	5.71	32.41
+	+	Sequential	5.33	5.81	32.52
+		Iterative	5.29	5.76	32.24
	+	Iterative	5.53	5.76	32.27
+	+	Iterative	5.27	5.79	32.22

Table 3. *Το μοντέλο transformer εκπαιδεύεται με τρεις διακριτούς τρόπους: όλο μαζί (Jointly), με επαναληπτική εκπαίδευση και με σειριακή εκπαίδευση. Η μέθοδος CAH εφαρμόζεται είτε σε έναν από τους μηχανισμούς αυτο-προσοχής είτε και στους δύο ταυτόχρονα. Αναφέρεται η μετρική perplexity στο σύνολο δεδομένων εκπαίδευσης (PPL Training) και στο σύνολο δεδομένων ελέγχου (PPL Test) και η μέτρηση BLEU στο σύνολο δεδομένων ελέγχου. Το μοντέλο εκπαιδεύτηκε για 15 εποχές.*

0.5.2 Εφαρμογή στο RoBERTa

Εκπαιδεύουμε το μοντέλο RoBERTa στο πρόβλημα της μοντελοποίησης γλώσσας πάνω σε κείμενο από την αγγλική έκδοση της Wikipedia. Εφαρμόζουμε την μέθοδο ΑΚΠ στους μηχανισμούς αυτό-προσοχής όλων των επιπέδων του μοντέλου. Αναφέρουμε μετά την έβδομη εποχή εκπαίδευσης τιμή της μετρικής perplexity στα δεδομένα εκπαίδευσης ίση με 12.55 για το μοντέλο RoBERTa και 12.76 για την έκδοση με χρήση της μεθόδου ΑΚΠ. Οι αντίστοιχες μετρικές για τα δεδομένα επαλήθευσης (validation set) είναι 10.04 και 10.15 αντίστοιχα. Συνεπώς ούτε η εφαρμογή της μεθόδου στο RoBERTa φαίνεται να βελτιώνει τις αποδόσεις του μοντέλου.

0.6 Συζήτηση και Πιθανές Λύσεις

Μετά από περαιτέρω έρευνα στο μοντέλο transformer ανακαλύψαμε ότι οι νόρμες των σημάτων ακυρώνουν σε ένα βαθμό τις τιμές συμβατότητας που ανατίθενται από τον μηχανισμό ΑΚΠ. Αυτό σημαίνει ότι το μοντέλο προσαρμόζεται στην μετατροπή που πραγματοποιούμε. Μία πιθανή λύση στο πρόβλημα είναι η αντικατάσταση του μηχανισμού από έναν που θα ενεργοποιεί τις πιο σχετικές κεφαλές και θα απενεργοποιεί τις λιγότερο σχετικές, στην λογική του μοντέλου των Goyal et al. (2019) [2]. Ωστόσο προκαταρκτικά πειράματα σε αυτή την κατεύθυνση έδειξαν πτώση της απόδοσης. Όπως ισχύει συνήθως με τις επαγωγικές προτιμήσεις, η υλοποίηση και ενσωμάτωσή τους στα μοντέλα είναι αρκετά δύσκολο να επιτευχθεί. Παρόλα αυτά, θεωρούμε ότι ένας μηχανισμός που θα οδηγεί στην εξειδίκευση των κεφαλών προσοχής θα μπορούσε να φανεί επωφελής και η λύση ίσως να βρίσκεται στην επιλεκτική ενεργοποίηση υποσυνόλου των κεφαλών.

Κάποιος θα μπορούσε να χρησιμοποιήσει άλλη μετρική εκτίμησης της σχετικότητας της κεφαλής στη θέση του εσωτερικού γινομένου με τα διανύσματα συνάρτησης, όπως την κατανομή των τιμών προσοχής (attention maps). Ακόμα κάποιος θα μπορούσε να δοκιμάσει να αντικαταστάσει τη ανταγωνιστική διαδικασία με μία μέθοδο κανονικοποίησης (regularization) που θα ωθεί τις κεφαλές στην εξειδίκευση.

Chapter 1

Introduction

1.1 The Problem of Creating Truly Intelligent Agents With Machine Learning

The term **Artificial Intelligence (AI)** is a very broad one. As will be noted in chapter 2 there are various approaches to defining it. Generally speaking, the goal of AI is to build agents that are able to solve a wide variety of real-world problems for humans which currently demand the active involvement of one or more people in order to be solved.

Machine Learning (ML) is considered to be a sub-field of AI, preoccupied with designing agents that are able to apply knowledge learned from prior experience to solve problems that are presented to them. Modern state-of-the-art (sota) machine learning methods employ forms of **pattern matching**, i.e. they detect patterns and draw conclusions about them using a set of data, called **training data**, which plays the role of prior experience; when presented with new samples they attempt to infer which of the learned patterns are present in these samples in order to apply their conclusions to them.

An example that is commonly employed to describe how ML algorithms function is house pricing. The price of a house depends on a number of factors, called **features**, including the area where the house is located, its square footage, the number of bedrooms, the year of construction, etc. ML models are given a training set of pairs of such house features along with the corresponding prices and are called to learn a mapping from the first ones to the latter. The desired outcome of this process is a model that is able to predict prices of houses it has never seen before. This is done through an interpolation process. Essentially, the model employs already seen house-price mappings to deduce reasonable prices for combinations of feature values it hasn't encountered before.

Obviously it is impossible to obtain a dataset covering every possible combination of feature values, thus the need for interpolation. If a new house presented to the model is similar to other houses, that the model has seen during training, the model will likely base its predictions on the these houses' prices, and will pay less attention to houses of the training set whose features differ a lot from the new one's. But, one could potentially want to use the model for predicting the prices of houses with feature values that are uncommon or even have never appeared before, like in the case of a house that is located at an area in which no houses of the training set were found. A human agent who is relatively familiar with this area, but not with prices of houses located at it, could do this easily, whereas modern ML algorithms would require the gathering of a new set of data about house prices in this new area and the re-training of the model using this data set. The gathering of a new set of samples is known to be tiresome and expensive because machine learning models generally need a lot of data for their training.

A truly intelligent model would be able to reason about the effect of locations on house prices

as well as how this is related to knowledge about the new location that may be readily available. Ideally, the model should connect the available pieces of information to infer a reasonable price or decide which of them are missing and either generate a prediction while also issuing a related warning regarding its low confidence or actively seek to fill in its knowledge gaps.

Modern ML models are unable to perform such reasoning processes that require a form of systematic thinking. As a result, where humans only need a handful of data to solve complicated problems, ML models require data sets of millions of samples. Solving this problem would bring the research community a step closer to what is known as **artificial general intelligence (AGI)**, a property possessed by thinking agents that are able to perform comparably to a human being in any possible task.

Solving this problem demands modifications in the way ML models store, process and combine pieces of knowledge as well as in their training processes. As Goyal and Bengio (2020) [51] note, models must be able to store knowledge in a way that allows them to decide on the fly which pieces of knowledge are relevant in a given situation and how these must be recomposed to enable the model to deal with the problem at hand. In addition to that, during training models must confront problems whose solution requires the development of the above skills on the part of the models.

1.2 Research Contributions

The goal of this thesis is twofold: to thoroughly present the work of a group of researchers lead by Yoshua Bengio, which aims to address the aforementioned problems, and to attempt to contribute to solving the problem of organized knowledge storage in the weights of a family of models that are based on the famous machine learning model called the **transformer** [1].

In more detail, we review parts of the work of Bengio and his research team in the direction of building models that are capable of performing the higher cognitive functions humans do. Their research comprises of a set of suggested modifications in neural architectures and training methods which are based on assumptions or else preferences, which they name **inductive biases**.

We also take a closer look at the sources of inspiration of these research efforts, which include notions from cognitive science, neuroscience and the study of causality.

Goyal and Bengio (2020) [51] contend that intervening in the way knowledge is organized inside a model can lead to more efficient models with improved generalization abilities. We thus propose two modifications to the well-known Transformer architecture, seeking to control the knowledge storing process in a way that results in the segmentation of the models into specialized modules. Specifically, we propose **a)** the replacement of the FFNN of each Transformer layer with a set of parallel FFNNs that are trained selectively based on a competition process and **b)** employing a similar competitive training framework to train the attention heads of the model. We implement the second idea using a softmax layer to promote competition among the attention heads. We analyze the results and discuss possible directions of future work.

1.3 Thesis Outline

In chapter 2, Machine Learning and Neural Networks, we attempt to define artificial intelligence (AI) and we also discuss related research areas in more detail. We also introduce fundamental machine learning (ML) concepts and algorithms. We further delve into the matter of neural networks (NNs) examining some core concepts and most basic neural network architectures.

In chapter 3, Natural Language Processing, we briefly present key aspects of the field of natural language processing (NLP), discussing some of its signature problems and focusing on the matter of language modeling (LM). We explain how modern neural networks are currently used to solve

NLP problems and examine two of the most widely used neural models in the field right now, the transformer and BERT.

In chapter 4, Modeling System 2 with Neural Networks, we start with a discussion about the sources of inspiration of the research efforts we present later in the chapter. We give the definitions of System 1 and System 2, as presented by Daniel Kahneman, we analyze the Global Workspace Theory introduced by Baart, provide examples as to how the notion of module specialization applies to the human brain and explain how the field of causality introduces the assumption of independent mechanisms. Following that, the approach of meta-learning is discussed. We then present the inductive biases suggested by Goyal and Bengio (2020) [51] and continue with their efforts to incorporate them to the methods used to train neural networks as well as to neural models' architectures.

In chapter 5, Transformers with Competitive Attention Heads and FFNNs, we point out problems that result from the choice to segment neural architectures in an effort to implement the notion of modularity. Consequently, we propose two modifications to transformer-based architectures: competitive FFNNs and competitive attention heads (CAHs). We implement the latter and analyze the results.

In chapter 6, Conclusions, we present our conclusions and suggest possible directions for future work.

Chapter 2

Machine Learning and Neural Networks

2.1 Introduction

Understanding human intelligence has always been one of mankind's biggest pursuits. From the time of Plato and the differentiation between discursive reason and intuitive reason [54] to the work of modern neuroscientists who, with the use of tools such as functional magnetic resonance imaging (fMRI) [55], study how the actual neurons of human brain work and cooperate to produce a thought or react to a certain stimuli. Yet, it is only 79 years ago, in 1943, that McCulloch and Pitts proposed the first model of an artificial neuron [56] and only 72 years ago, in 1950, that Alan Turing spoke of "thinking machines" [57], setting the foundations of Artificial Intelligence (AI). Despite the shortness of this period humanity has produced chat-bots that are today used in costumer service, banking and even healthcare; it has created models that perform comparably to humans in certain image classification tasks [58] and has trained agents that can beat human experts at complex games such as Go [59] and Stratego [60]. Therefore, the public's excitement about Artificial Intelligence and Machine Learning (ML) is understood, but before one delves into the exciting world of Artificial Intelligence, an acquaintance with some basic terms is necessary.

Defining **Artificial Intelligence** is not that easy. As noted by two AI pioneers, Artificial Intelligence research comes in a variety of forms, depending on the degree rationality that the agent is expected to exhibit and the mechanism that produces the desirable intelligent behavior [61]. Next the 4 combinations that result from these two dimensions will be discussed, and during this discussion definitions of additional useful terms will be given, in a way that the importance of the corresponding areas of research to the umbrella field of Artificial Intelligence is best understood.

- **Acting Humanly**: In his 1950 article [57], Alan Turing proposed the Turing test as a method to answer to the question of whether a certain "machine can think". A computer is asked written questions by a human interrogator, and passes the test if, upon providing the human with answers, the human interrogator can't tell if these were given by another human or a computer. In this version of Artificial Intelligence, the machine's answers should not be mathematically perfect, but simply logical in the same way a human's answers would be expected to be. Moreover, in this version of Artificial Intelligence, what is tested is not the internal process that the machine performs but rather its behavior.

  Russel and Norvig note that the machine would need the following capabilities to pass the test:

  - **Natural Language Processing**: a field of Artificial Intelligence that is concerned with enabling computers to analyze and understand written and spoken human language, and also to respond using written and spoken human language [62]. Natural Language Processing (NLP) will be further discussed in chapter 3.

  - **Knowledge Representation**: a field of Artificial Intelligence that is concerned with studying ways to represent information about the real world in a form that enables computers to reason with [62].
  - **Automated Reasoning**: a field of Artificial Intelligence that is concerned with enabling computers to manipulate knowledge by making logical inferences towards a certain goal that can be either provided by the user or decided by the computer itself [62].
  - **Machine Learning**: a field of Artificial Intelligence that is concerned with enabling computers to learn from experience. Learning from observations means that the system should possess the skills to find patterns in the data, assess their significance and integrate the newly gained knowledge successfully [62, 63]. Machine Learning subject areas include computer science, mathematics, statistics, Data Mining, Deep Learning, data science and NLP [64]. Machine Learning and Deep Learning will be further discussed in the following chapters.

  A total Turing test would require a physical entity, a robot, that can interact with real world objects and people. In that case the following would also be required:

  - **Computer Vision**: a field of Artificial Intelligence that deals with how computers can derive useful high-level information from visual inputs such as images and videos [65].
  - **Robotics**: an interdisciplinary branch of computer science and engineering with a goal of creating machines, called robots, that can be programmed as to carry out a large set of orders. It is important to note that if the machine can only deal with a narrow order category then it is probably not a robot [66].

  Yet most researchers believe that studying the fundamentals of intelligence, and not passing the Turing test, is the way to AI. Therefore they shift from trying to achieve intelligent behavior to attempting to program the internal processes of thinking.

- **Thinking Humanly**: Researchers pursuing this goal strive to first comprehend the way that humans think. Russel and Norvig provide three tools that researchers use to achieve this goal: **introspection**, **psychological experiments** and **brain imaging**. Using these tools researchers can develop theories about the inner working of the human mind and then test them by analyzing the behavior of computer programs that implement them.

  - **Cognitive Science**: the study of the mind processes. It is directly connected to the fields of neuroscience, psychology, computer science and others. This term will be further developed in the following chapters as ideas emanating from cognitive science have found and continue to find applications to AI models.

  But trying to imitate the human way of thinking is not the only way to create intelligence. Humans rarely do mathematical calculations before making a decision, but no one would call a machine dumb for doing so.

- **Thinking Rationally**: Logical rules are a part of the field called **Logic** and were first introduced by the ancient Greek philosopher Aristotle. The rules, given correct inputs - in the sense of true in the real world - in the form of logical sentences are bound to yield correct outputs or conclusions. Researchers of this area, called Logicists, try to use Logic to create Artificial Intelligence. But, as it is impossible to be certain of the set of rules that govern a system and the validity of the input, the theory of **probability** is often used to deal with this uncertainty.

The last version of Artificial Intelligence is concerned with enabling the machine to act rationally, aside from thinking rationally.

- **Acting Rationally**: A machine acting rationally is called a rational agent and it is expected to pursue through its actions the best possible outcome. This version is the one that has been pursued most by AI researchers in the field's history because of its advantages over the other versions:
  - **Generality**: It is more general than the "rules of thought" approach as correct inference is just one of the capabilities a rational agent should possess. It also uses all the subfields used by the first approach as it is expected to find solutions to real world problems, implement them and communicate with other agents and humans.
  - **Mathematical soundness**: It is mathematically well defined and that assists the design of machines that provably achieve it. Th same is not true with the first two versions that deal with the human mind and behavior.

2.2 Machine Learning Types

As the main goal of Machine Learning is to learn from experience it is evident that any Machine Learning process is based on real world observations. These observations can be images, videos, written or spoken language, numbers etc. Machine Learning researchers have concluded that many Machine Learning algorithms work best when the raw real world observations are first processed in a way that makes it easier for the algorithms to find patterns in the data. This procedure is called **preprocessing** and it can vary from being as simple as removing punctuation from raw text to being very complicated such as isolating different objects from one another in an image. Another optional step that may come next is using the preprocessed data to extract useful features, such as the number of appearances of a word in a text or the length of an object in a picture. This step is called **feature extraction**. The resulting informative units are called **feature vectors** [3].

Machine Learning categories differ from each in the type of information that accompanies these data points.

- **Supervised Learning**: In a Supervised Learning setting the Machine Learning model is provided with a **label** for each feature vector. The goal of supervised Machine Learning algorithms is to exploit the information provided by the given vector – label pairs, called **samples**, in order to be able to correctly predict the labels of unseen feature vectors. This ability is called **Generalization**. In that sense what is learned is a function that maps feature vectors to labels. These pairs constitute the **training set** of the problem. In order to evaluate the model a number of feature vectors, for which the correct labels are know a priori, are given as inputs to the model and its possibly wrong outputs are compared to the correct labels. This set of pairs is called the **test set** of the problem.
  - Labels can be categorical, like the topic of a text or an object shown in a picture. In this case the set of possible labels is a finite one and the problem is called a **classification** problem [4].
  - Labels can also be sets of one or more continuous variables, like the price of a house or of a certain stock in the stock market. In that case the problem is called a **regression** problem [4].
- **Unsupervised Learning**: In an Unsupervised Learning setting the Machine Learning model is only provided with the feature vectors and not the corresponding labels. The goal in

this setting is to group these feature vectors by some measure of similarity that is defined explicitly or implicitly in the chosen model itself. The formed groups are called clusters and the corresponding process is also called **clustering**. While all feature vectors that belong in the same cluster must share some similarities, different algorithms may yield different clusters for the same set of feature vectors. In a **hard clustering** process a feature vectors may belong to only one cluster while in a **soft clustering** one many clusters may share a feature vector.

A clustering algorithm may be used for example by a marketing team to discover the potential client subgroups, examine their traits and design their products and marketing campaigns accordingly. Another possible use of a clustering algorithm is to improve the understanding of the structure of a certain dataset and then use this knowledge build a supervised learning model that is better suited to it [67].

- **Semi-supervised Learning**: Labeling samples is usually performed manually and can be very costly to do so. Therefore some datasets only have a, usually small [68], portion of their samples labeled, while this kind of information does not exist for the rest of the points and they are consequently called **unlabeled**. The goal in such a setting can be similar to the supervised setting's one. In that case the unlabeled samples provide information about the nature of the data, that might not be recoverable from the labeled points. This knowledge can then be used in the design of the supervised learning model.

  It is also possible to use this kind of information to perform clustering. For example, in the case of categorical labels, the user can impose a constraint on the clustering algorithm, that labeled samples of the same category must belong to the same cluster while samples of different categories should belong to different clusters [67].

- **Reinforcement Learning**: In some real world scenarios the possible there may innumerable different positions an agent may find itself into. These problems are impossible to solve using supervised learning, that uses labeled samples to extrapolate to generalize to unseen cases. Even in the case of a simple chess match there are about 1040 possible chess positions, rendering supervised learning inconvenient due to the huge size of necessary position – move example pairs, provided by a chess-master for example. If a machine finds itself in a similar scenario it cannot afford to passively wait for the needed information like in the supervised learning setting, but it should actively explore the search space to obtain effective tactics of achieving its goal. Instead of expensive labels the computer needs only to be given a response to its actions, informing it whether it has achieved its goal or not, which can be easily programmed because goals are usually well defined. The positions that computer may find themselves in are called **states**, a move they may choose to perform is called an **action**, the sets of state – action pairs that computers choose from are called **policies** and the system's responses to machines' actions are called **rewards**. The computer's target is to find a policy that maximizes the expected sum of rewards, starting from a given initial state. If the computer is only rewarded for achieving its goal, like in chess, then the rewarding system is called **sparse**. However the are games like tennis in which rewards may be given even before the goal is achieved, like when a points or a set is won. What is very interesting is that a computer may occasionally choose to temporarily receive a smaller reward in order to explore state – action combinations that may yield larger rewards in the future [61, 69].

2.3 Machine Learning Concepts and Models

At the foundations of Machine Learning lie certain concepts that provide the mathematical justification of Machine Learning algorithms. In this chapter, basic Machine Learning concepts will be discussed as well as algorithms that directly result from them.

2.3.1 Bayesian Decision Theory

The training set of a classification problem is composed of N samples, comprised of feature vectors along with the respective labels. The feature vector of the i-th sample is represented by $\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}, \cdots, x_D^{(i)}]^T$ and belongs to a class $\omega^{(i)}$. The set of all feature vectors of the dataset is represented by the notation $\{\mathbf{x}\}_1^N$ and the finite set of possible classes is represented by $\Omega = \{\boldsymbol{\omega}\}_1^M$. The following statistical quantities are of interest:

- The probability of appearance of a class i is equal to $\mathrm{P}(\omega_i)$ and is called **a priori probability**. A priori probabilities are usually estimated using the rate of appearance of each class in the dataset. For example $P(\omega_i) \approx N_i/N$, where N_i is the number of samples that belong to the i-th class. Assuming that N is much larger than the number of classes this estimation is quite accurate.

- The conditional probability density function of a feature vector $\mathbf{x}$ belonging to the i-th class, $p(\mathbf{x}|\omega_i)$, is called **class-conditional probability density function**.

- The conditional probability density function of a feature vector $\mathbf{x}$ belonging to a class ω_i, $p(\omega_i|\mathbf{x})$, is called **a posteriori probability**.

- It is generally assumed that all feature vectors are drawn from a pdf p: $\mathbf{x}^{(i)} \sim p, i \in \{1, ..., N\}$, which is called **evidence probability**.

Bayes rule connects the above quantities with the equation:

$$p(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})}, \text{ where} \tag{2.1}$$

$$p(\mathbf{x}) = \sum_{i=1}^{M} p(\mathbf{x}|\omega_i)P(\omega_i) \tag{2.2}$$

In the case of a set of two possible classes, ω_1 and ω_2, the **Bayesian classification rule** indicates that:

- if $p(\omega_1|\mathbf{x}) > p(\omega_2|\mathbf{x})$, then $\mathbf{x}$ is classified as belonging to the first class, ω_1

- if $p(\omega_1|\mathbf{x}) < p(\omega_2|\mathbf{x})$, then $\mathbf{x}$ is classified as belonging to the second class, ω_2

In the case of equality the sample can be assigned to either of the two classes. Using equation 2.1, the inequalities can be written as

$$\frac{p(\mathbf{x}|\omega_1)P(\omega_1)}{p(\mathbf{x})} \lessgtr \frac{p(\mathbf{x}|\omega_2)P(\omega_2)}{p(\mathbf{x})}$$

and by cancelling out $p(\mathbf{x})$, which does not depend on the class identity, one can decide about the value of the label of $\mathbf{x}$ without computing the evidence probability:

$$p(\mathbf{x}|\omega_1)P(\omega_1) \lessgtr p(\mathbf{x}|\omega_2)P(\omega_2) \tag{2.3}$$

By writing down the total classification error and computing the areas under the curve it is proven that the total classification error is indeed minimized at the point suggested by equation 2.3.

$$P_e = P(\mathbf{x} \in R_2, \omega_1) + P(\mathbf{x} \in R_1, \omega_2) \Rightarrow$$

$$P_e = P(\mathbf{x} \in R_2|\omega_1)P(\omega_1) + P(\mathbf{x} \in R_1|\omega_2)P(\omega_2) \Rightarrow$$

$$P_e = \int_{R_2} p(\mathbf{x}|\omega_1)P(\omega_1)\, dx + \int_{R_1} p(\mathbf{x}|\omega_2)P(\omega_2)\, dx \tag{2.4}$$

This is explained graphically in figure 2.1, where it is shown that following the Bayesian classification rule yields the minimum possible expected classification error [3].

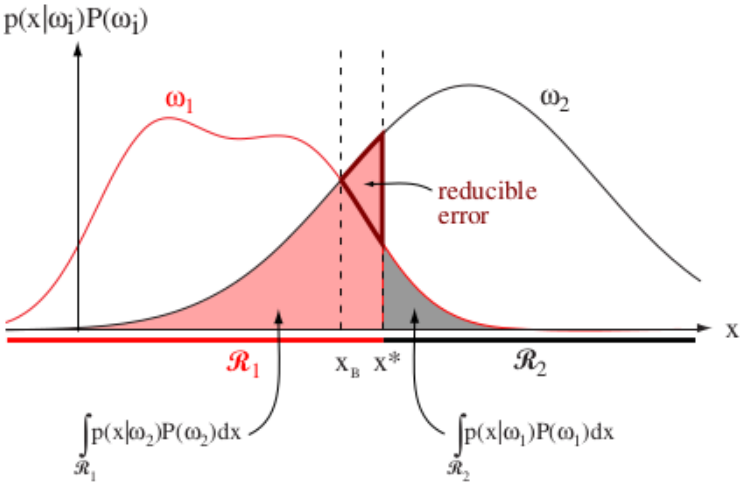


Figure 2.1. *The expected error is reduced when the chosen point is the one that minimizes the sum of the two areas under the curves. This point is the one chosen by the Bayesian classification rule, x_B, while choosing any other point x^* yields additional error [3].*

It can also be shown, by maximizing the probability of being correct, that in the case of M classes assigning $\mathbf{x}$ to ω_i if $p(\mathbf{x}|\omega_i)P(\omega_i) > p(\mathbf{x}|\omega_j)P(\omega_j), \forall j \neq i$, also minimizes the classification error probability.

Viewing the matter from a different angle it is evident that, if the class-conditional and the a priori probabilities are known, then the Bayesian classification error is the minimum possible error that can be achieved and it is achieved using the Bayesian classification rule [67].

2.3.2 Discriminant Functions

If the feature vector $\mathbf{x}$ is viewed as a continuous function then the feature space can be partitioned into M regions, where M is the number of possible classes. Between two contiguous regions, R_i and R_j, there lies a surface that is defined by the equation $p(\mathbf{x}|\omega_i)P(\omega_i) = p(\mathbf{x}|\omega_j)P(\omega_j)$. This surface is called a **decision surface**. So far only probabilistic interpretations of the decision process have been used. Yet one can also use any monotonically increasing function f to create functions of the form $g_i(\mathbf{x}) = f(p(\omega_i|\mathbf{x}))$. Functions of the form of g_i are called **discriminant functions** and they can be used to assign a feature vector $\mathbf{x}$ to a class i if it is true that $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for every other class j [67].

The probabilistic approach described in 2.3.1 is well suited to problems in which modeling class-conditional probability density functions is possible. Models that result from the computation of $p(\mathbf{x}|\omega_i)$ are called **generative models** as it is possible to use the class-conditional probability to generate instances of the class ω_i. Nevertheless, if the class-conditional probability density

functions are too complicated to estimate explicitly, one might choose instead to model discriminant functions, creating models that are known as **discriminative models**.

2.3.3 Maximum Likelihood Estimation

Usually, the class-conditional probability is unknown and it must be modelled before proceeding to the computation the two sides of the aforementioned inequalities. One can approach this modeling problem in various ways. One way that will be discussed in this chapter, is called **Maximum Likelihood Estimation (MLE)** . In the general case of an M-class classification problem with likelihood functions $p(\mathbf{x}|\omega_i), i \in 1, \cdots, M$ the following assumptions are made:

i) **Parametric form**: The likelihood functions are assumed to be of a parametric form with the parameter vectors $\boldsymbol{\vartheta}_i$, $i \in 1, \cdots, M$, being unknown

ii) **Functional independence**: The estimation of a parameter vector $\boldsymbol{\vartheta}_i$ of a certain likelihood function $p(\mathbf{x}|\omega_i)$ is independent from the estimation of every other parameter vector $\boldsymbol{\vartheta}_j, j \neq i$

iii) **i.i.d. feature vectors**: Different feature vectors are statistically independent and identically distributed

Due to the first assumption the likelihood of a certain class can written as $p(\mathbf{x}|\omega_i; \boldsymbol{\vartheta}_i)$. This assumption essentially means that a likelihood function is considered to belong to a distribution family and is made specific with the determination of the parameter vector. Owing to the second assumption the parameters of each class can be computed independently from the parameters of the other classes. Then one can focus on random samples $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(N_i)}$ of the same class i that are drawn from $p(\mathbf{x}|\omega_i; \boldsymbol{\vartheta}_i)$ and their joint pdf $p(X_i|\omega_i; \boldsymbol{\vartheta}_i)$, which will be represented by $p(\mathbf{x}|\boldsymbol{\vartheta}_i)$ and $p(X_i|\boldsymbol{\vartheta}_i)$ respectively from now on. Because of the third assumption it is true that

$$p(X_i|\boldsymbol{\vartheta}_i) = p(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots, \mathbf{x}^{(N_i)}|\boldsymbol{\vartheta}_i) = \prod_{k=1}^{N_i} p(\mathbf{x}^{(k)}|\boldsymbol{\vartheta}_i) \tag{2.5}$$

Solving the problem, in general, means estimating $\boldsymbol{\vartheta}_i$. Using Bayes theorem:

$$\hat{\boldsymbol{\vartheta}}_i = arg\ \max_{\boldsymbol{\vartheta}_i} p(\boldsymbol{\vartheta}_i|X_i) = arg\ \max_{\boldsymbol{\vartheta}_i} \frac{p(X_i|\boldsymbol{\vartheta}_i)p(\boldsymbol{\vartheta}_i)}{p(X_i)} = arg\ \max_{\boldsymbol{\vartheta}_i} p(X|\boldsymbol{\vartheta}_i)p(\boldsymbol{\vartheta}_i) \tag{2.6}$$

In Maximum Likelihood Estimation the parameter vectors are computed as to maximize $p(X_i|\boldsymbol{\vartheta}_i)$. This is equivalent to choosing, out of all the members of the family of probability distributions that the likelihood function belongs to, the one that better explains that data at hand. Then, dropping the class-specific notation, it is true that:

$$\hat{\boldsymbol{\vartheta}}_{MLE} = arg\ \max_{\boldsymbol{\vartheta}} \prod_{k=1}^{N} p(\mathbf{x}^{(k)}|\boldsymbol{\vartheta}) \tag{2.7}$$

By applying the logarithm function to the product in equation 2.7 the **log-likelihood** is defined:

$$L(\boldsymbol{\vartheta}) = \ln \prod_{k=1}^{N} p(\mathbf{x}^{(k)}|\boldsymbol{\vartheta}) \tag{2.8}$$

At $\hat{\boldsymbol{\vartheta}}_{MLE}$, the likelihood function's gradient w.r.t. $\boldsymbol{\vartheta}$ must then be equal to zero. Since the logarithm is a monotonic function, the equality of the derivative to zero is also applied to equation

2.8. Then, using the logarithm's properties results to:

$$\frac{\partial L(\boldsymbol{\vartheta})}{\partial \boldsymbol{\vartheta}} = \sum_{k=1}^{N} \frac{\partial \ln(p(\mathbf{x}^{(k)}|\boldsymbol{\vartheta})}{\partial \boldsymbol{\vartheta}} = \sum_{k=1}^{N} \frac{1}{p(\mathbf{x}^{(k)}|\boldsymbol{\vartheta})} \frac{\partial p(\mathbf{x}^{(k)}|\boldsymbol{\vartheta})}{\partial \boldsymbol{\vartheta}} = 0 \tag{2.9}$$

It can be shown that, under some conditions that are true under most circumstances, the estimate converges in the mean to the true value of $\boldsymbol{\vartheta}$, meaning that it is **asymptotically unbiased**. Note that the parameters must be computed for every class separately [67].

As was previously stated, the Bayesian classification error is the minimum possible error that can be achieved by a classifier. But the total classification error is almost always larger than that. Failing to include a true likelihood function in the family of distributions expressed by the assumed model inevitably leads to a modeling error. Moreover, even if the true likelihood function is indeed included in the family of distributions assumed, one has to account for possible estimation errors that usually occur because of the limited number of available examples.

2.3.4 Maximum a Posteriori Estimation

The difference between MLE and **maximum a posteriori estimation (MAP)** is that, while in the first case $\boldsymbol{\vartheta}$ is considered to be an unknown parameter, in the latter case it is considered to be a random vector. Then, equation 2.7 becomes:

$$\hat{\boldsymbol{\vartheta}}_{MAP} = arg\, \max_{\boldsymbol{\vartheta}} \prod_{k=1}^{N} p(\mathbf{x}^{(k)}|\boldsymbol{\vartheta}) p(\boldsymbol{\vartheta}) \tag{2.10}$$

with the only difference to equation 2.8 being the existence of p($\boldsymbol{\vartheta}$) in the relationship. This is equivalent to adding bias to the estimation of $\boldsymbol{\vartheta}$. If knowledge about $\boldsymbol{\vartheta}$ exists then with MAP it can be integrated to the computations. Additionally, if $\boldsymbol{\vartheta}$ is assumed to be a uniform distribution, in which case all values are equally probable, this method becomes equivalent to the MLE [67].

2.3.5 Linear Regression

Regression problems are a category of supervised learning problems. The output in this case is a continuous variable and the problem is to find the relation between the input variables, which constitute the feature vector, and the output variable. The simplest form of a regression model is the discriminative model in which the output variable is the sum of a linear combination of the input variables plus a constant. This is called **linear regression** (LR) and, by introducing an intercept term, it can be written as:

$$y(\mathbf{x}, \boldsymbol{\vartheta}) = \theta_0 + \theta_1 x_1 + \cdots + \theta_D x_D = \sum_{d=0}^{D} \theta_d x_d = \boldsymbol{\vartheta}^T \mathbf{x}, \text{ where } x_0 = 1 \tag{2.11}$$

The parameters θ_0 and $\theta_1, \theta_2, \cdots, \theta_D$ are also called **bias** and **weights** respectively. The training set of a regression problem is comprised of pairs $\{\mathbf{x}^{(i)}, y^{(i)}\}_1^N$, where $\mathbf{x}^{(i)}$ is the feature vector of the i-th sample and $y^{(i)}$ is the corresponding label. To perform the task of fitting the model $y(\mathbf{x}, \boldsymbol{\vartheta})$ to the data $\{\mathbf{x}^{(i)}, y^{(i)}\}_1^N$, a **cost function**, also called a **loss function**, is defined:

$$J(\boldsymbol{\vartheta}) = \frac{1}{2} \sum_{k=1}^{N} (y(\mathbf{x}^{(k)}, \boldsymbol{\vartheta}) - y^{(k)})^2 \tag{2.12}$$

The cost function measures "how good" $y(\mathbf{x}, \boldsymbol{\vartheta})$ models the training set. Obviously, the smaller the cost function is, the better $y(\mathbf{x}, \boldsymbol{\vartheta})$ is at modeling the training set. It is then a natural next

step to try estimating $\boldsymbol{\vartheta}$ by minimizing the cost function. This introduces an important topic, which is the optimization of a cost function, and in this particular case the minimization of the **mean square error (MSE)**. Even though equating the derivative of the cost function to zero and solving the resulting system of equations can be done in this case, as the the system is linear w.r.t. the unknown parameters, doing so is not always possible because of the complexity of most cost functions. One can alternatively begin by guessing an initial value for $\boldsymbol{\vartheta}$ and then start taking small steps towards the direction that minimizes the cost function. In the case that the direction of these steps is determined by the derivative of the cost function with respect to $\boldsymbol{\vartheta}$ the optimization algorithm is called **gradient descent (GD)**:

ALGORITHM 2.1: *Gradient Descent*

initialize $\boldsymbol{\vartheta}$ with $\boldsymbol{\vartheta}_0$
$i \leftarrow 0$
while algorithm has not converged **do**
  $\boldsymbol{\vartheta}_{i+1} = \boldsymbol{\vartheta}_i - a\nabla J(\boldsymbol{\vartheta}_i)$
  $i \leftarrow i + 1$
end while

The minus sign in the update rule indicates that a step is taken as to minimize $J(\boldsymbol{\vartheta}_i)$. The scalar value a is called **learning rate** and it determines the size of the step taken. Convergence rules are decided depending on the problem. Possible convergence conditions could be the reduction of the value of the cost function or of the rate of change of the parameter value below a threshold.

In the case of linear regression the gradient descent update rule is:

$$\boldsymbol{\vartheta}_{i+1} = \boldsymbol{\vartheta}_i - a\sum_{k=1}^{N}(\boldsymbol{\vartheta}_i^T\mathbf{x}^{(k)} - y^{(k)})\mathbf{x}^{(k)} \tag{2.13}$$

Intuitively, what happens is that the error for each sample is computed and then multiplied by the corresponding feature vector. If the error is negative then the parameter vector is moved closer to the feature vector, so that their inner product yields a larger output. If the error is positive then the opposite happens [4].

Linear Regression and Maximum Likelihood

Linear regression and maximum likelihood are connected in a very interesting way. Assume that:

i) The training samples are statistically independent and identically distributed (**iid**)

ii) The labels are the **response** of a **stationary system** to the input variables, which are the feature vectors

iii) The uncertainty over the value of a label t is expressed using a probability distribution. Uncertainties may arise from an inability to consider all factors that affect the output variable, that is from missing some of the inputs, or from the addition of random noise. Given $\mathbf{x}$, t is assumed to follow a **Gaussian** distribution with a mean value of $y(\mathbf{x}, \boldsymbol{\vartheta})$. This is otherwise stated as

$$p(t|\mathbf{x}; \boldsymbol{\vartheta}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \boldsymbol{\vartheta}), \beta^{-1}) \tag{2.14}$$

or

$$t = y(\mathbf{x}, \boldsymbol{\vartheta}) + \varepsilon, \text{ where } p(\varepsilon) = \mathcal{N}(\varepsilon|0, \beta^{-1}) \tag{2.15}$$

The goal is again estimating the unknown parameters $\boldsymbol{\vartheta}$. Using the first assumption and

equation 2.14:

$$p(Y|\mathbf{x};\boldsymbol{\vartheta},\beta)=\prod_{k=1}^{N}\mathcal{N}(y^{(k)}|y(\mathbf{x}^{(k)},\boldsymbol{\vartheta}),\beta^{-1}),\text{ where } Y=\{y^{(k)}\}_1^N \tag{2.16}$$

Then, after applying the logarithm and substituting for the form of the Gaussian distribution:

$$L(\boldsymbol{\vartheta})=\ln\, p(Y|\mathbf{x};\boldsymbol{\vartheta},\beta)=-\frac{\beta}{2}\sum_{k=1}^{N}(y(\mathbf{x}^{(k)},\boldsymbol{\vartheta})-y^{(k)})^2+\frac{N}{2}\ln\beta-\frac{N}{2}\ln(2\pi) \tag{2.17}$$

Since the first term is the only one that is dependent on $\boldsymbol{\vartheta}$, applying MLE and maximizing $L(\boldsymbol{\vartheta})$ means minimizing the first term which has the same roots with the cost function $J(\boldsymbol{\vartheta})$ of 2.12. Therefore MLE and MSE minimization yield the same result under the assumptions presented above [4].

Generalized Linear Regression

If the system generating the samples is indeed linear w.r.t. to $\mathbf{x}$ then using the method presented above one can sufficiently fit the data at hand. Yet, usually, this is not the case, but the output may instead be dependent on the input in a nonlinear manner. Fortunately, taking the derivative of $y(\mathbf{x},\boldsymbol{\vartheta})$ w.r.t. $\mathbf{x}$ is nowhere needed in the calculations and only its derivative w.r.t. $\boldsymbol{\vartheta}$ is taken in the optimization process. Therefore, one needs not be restricted to a linear dependence on the input, but can create nonlinear functions of it and then use a linear combination of them to model highly nonlinear systems. Such functions are called **basis functions** and they can be used as a part of a feature extraction process that was briefly mentioned in chapter 2.2. An example of this, in the case of a 3-dimensional input $\mathbf{x}$, is the feature vector $\boldsymbol{\varphi}(\mathbf{x})=[1,x_1,x_2,x_3,x_1x_2,x_1x_3,x_1^2,x_3^3]^T$. Note that the dimension of the feature vector $\boldsymbol{\varphi}(\mathbf{x})$ can be smaller or even bigger than the dimension of the input. With the use of non-linear feature vector equations 2.12 and 2.13 become:

$$J(\boldsymbol{\vartheta})=\frac{1}{2}\sum_{k=1}^{N}(y(\boldsymbol{\varphi}(\mathbf{x}^{(k)}),\boldsymbol{\vartheta})-y^{(k)})^2 \tag{2.18}$$

$$\boldsymbol{\vartheta}_{i+1}=\boldsymbol{\vartheta}_i-a\sum_{k=1}^{N}(\boldsymbol{\vartheta}_i^T\boldsymbol{\varphi}(\mathbf{x}^{(k)})-y^{(k)})\boldsymbol{\varphi}(\mathbf{x}^{(k)}) \tag{2.19}$$

Overfitting and Regularization

Manufacturing feature vectors enables the modeling of highly complex functions, but it also creates a new problem. What happens when the data present in the training set can be modeled by a number of complicated functions of the input, but the actual system responsible for producing the dataset is much simpler? Even worse, there are cases in which random noise may affect the values of the labels and drive the optimization algorithm into computing parameters that model this influence. An example of this is shown in figure 2.2.

Polynomials with $M=0$ (e.g. $y=\theta_0$) or $M=1$ (e.g. $y=\theta_1x+\theta_0$) are too simple to model the sine function. On the contrary, a polynomial with $M=9$ can fit the data perfectly, but again fails to capture the underline structure. Therefore providing a model with high degrees of freedom may lead to over-complicated assumptions, rendering it incapable of generalizing to unseen instances. This problem is known as **overfitting**. A better model would be a polynomial with $M=3$ as shown in the figure. But this model, in contrast to the one with $M=9$, does not yield zero error.

This examples shows that minimizing the error may itself not be enough to create a model

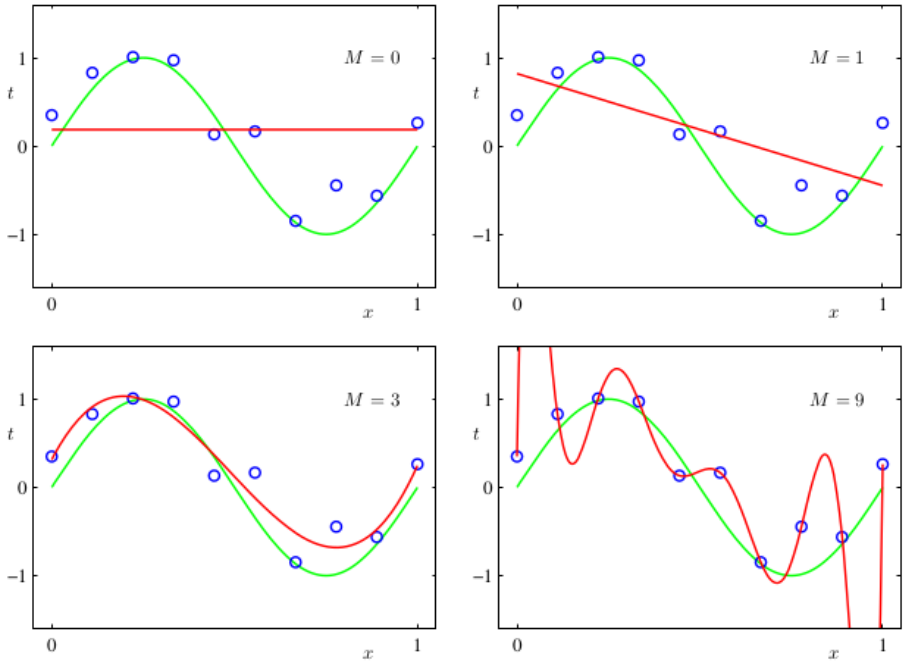


Figure 2.2. *Polynomial models of various orders M (red) modeling fitting data created by $sin(2\pi)$ (green) with the addition of Gaussian random noise [4]*

with generalizing capabilities. This is why another term, called **regularizer**, is usually added to the cost function, which helps to mitigate the aforementioned problem. This process is called **regularization**.

Coming back to the example, one might not know the ideal value for M beforehand and may use a larger value initially. A regularizer must then eliminate higher order coefficients while allowing the rest to model the unknown function. After the addition of a possible regularizer, the new cost function can be written as:

$$J(\vartheta) = J_{error}(\vartheta) + \frac{\lambda}{2}\sum_{k=1}^{D}\theta_k^2 \tag{2.20}$$

where D is the polynomial's order and the coefficient λ determines the relative importance of the regularization term with the error one. Minimizing this cost function will push both the error and the coefficients towards zero and, if λ is chosen correctly, only unnecessary coefficients will be eliminated, while the rest will be computed as to minimize the error term. Because this regularizer drives weights to zero it is known as **weight decay**. The parameter λ, as well as the order of the polynomial, M, and many others are known as **hyper-parameters**. They are chosen before the trainable model parameters are adjusted through a training process. As in the case of λ, where a delicate balance is required between error minimization and regularization, the values of hyper-parameters must be chosen carefully so that the resulting model architecture be ideal for the problem at hand. A method that is usually used to discover ideal values for hyper-parameters will be discussed in chapter 2.4.6.

Similarly, with the use of a regularizer, the cost function in equation 2.18 becomes:

$$J(\vartheta) = \frac{1}{2}\sum_{k=1}^{N}(y(\varphi(\mathbf{x}^{(k)}), \vartheta) - y^{(k)})^2 + \frac{\lambda}{2}\sum_{k=1}^{D}\theta_k^2 \tag{2.21}$$

This method is called **Regularized Least Squares** [4].

2.3.6 Logistic Regression

The concepts that were discussed in chapter 2.3.5 are also relevant to the classification problem. First consider a classification problem of two classes, ω_1 and ω_2, with a training set of N samples $\{\mathbf{x}^{(i)}, \omega^{(i)}\}_{i=1}^{N}$, where $\mathbf{x}^{(i)}$'s are vectors. Since there are two classes it seems natural to use a model that will assign a value of 0 to examples it believes to belong to the first class and a value of 1 to examples which it believes to belong to the second one. Without loss of generality, for this problem, 0 will correspond to ω_1 and 1 to ω_2. If the model is not sure about its decision then it could assign a value belonging in the interval (0, 1), which can be closer to 0 if it believes that the sample most likely belongs to class ω_1 and closer to 1 to signal the opposite.

Unfortunately this model cannot be the linear function used in chapter 2.3.5 as the linear function may assign any value to an input and is not limited to [0, 1]. For this reason the **sigmoid function** is used:

$$\sigma_{\boldsymbol{\vartheta}}(\mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\vartheta}^T \mathbf{x}}} \tag{2.22}$$

As can be seen in figure 2.3, the sigmoid function's values are limited to the interval (0, 1) and swiftly change from lower to higher values when a threshold, that is determined by the zeroth order coefficient of $\boldsymbol{\vartheta}$, is exceeded. This is a desired property for input values lying close to the decision surface, discussed in chapter 2.3.2. The rate of change is also determined by $\boldsymbol{\vartheta}$ as one can see in the figure below.

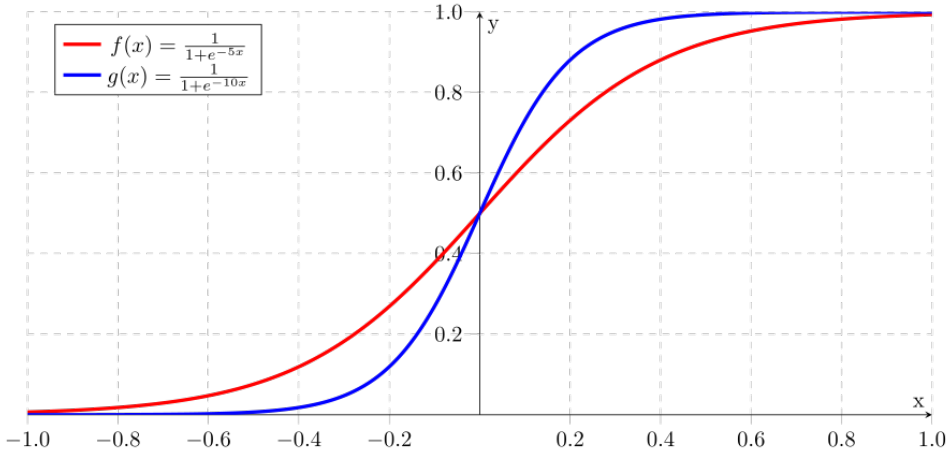


Figure 2.3. *Sigmoid function for 1-dimensional input and for two values of ϑ, ϑ=5 (red) and ϑ=10 (blue). Image from* `https://commons.wikimedia.org/wiki/File:Sigmoid-function.svg`

The sigmoid also has the very useful property, that its derivative can be expressed in terms of the sigmoid itself:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{2.23}$$

In order to compute the parameters of the linear regression model the mean squared error was used as a cost function. But the quadratic loss is symmetric w.r.t. the label value, which is intuitively wrong in the case of classification. Take for example the assignment to a sample, that belongs in the second class, of a value that is bigger than 1 by a positive scalar e and the assignment to the same sample of a value that is smaller than 1 by the same scalar value. They both would create the same error signal, even though the model would have made a better prediction in the first case.

In the previous chapter it was shown that, under three assumptions, applying the MLE to a linear regression model reveals the intuitively correct loss function. As will be shown below, this is also true for a logistic regression model, when a more suitable third assumption is used [4]. This

assumption lies in changing equation 2.14 to:

$$p(y = \omega_1|\mathbf{x}; \boldsymbol{\vartheta}) = 1 - \sigma(\mathbf{x}, \boldsymbol{\vartheta})$$

$$p(y = \omega_2|\mathbf{x}; \boldsymbol{\vartheta}) = \sigma(\mathbf{x}, \boldsymbol{\vartheta})$$

or in a more compact form:

$$p(y|\mathbf{x}; \boldsymbol{\vartheta}) = \sigma_{\boldsymbol{\vartheta}}^{y}(\mathbf{x})(1 - \sigma_{\boldsymbol{\vartheta}}(\mathbf{x}))^{(1-y)} \tag{2.24}$$

Next, the negative of the log-likelihood is computed:

$$\begin{aligned} L(\boldsymbol{\vartheta}) &= -\ln \prod_{k=1}^{N} p(y^{(k)}|\mathbf{x}^{(k)}; \boldsymbol{\vartheta}) \\ &= -\ln \prod_{k=1}^{N} (\sigma_{\boldsymbol{\vartheta}}(\mathbf{x}^{(k)}))^{y^{(k)}} (1 - \sigma_{\boldsymbol{\vartheta}}(\mathbf{x}^{(k)})^{(1-y^{(k)})} \\ &= -\sum_{k=1}^{N} (y^{(k)} \ln\, \sigma_{\boldsymbol{\vartheta}}(\mathbf{x}^{(k)}) + (1 - y^{(k)}) \ln(1 - \sigma_{\boldsymbol{\vartheta}}(\mathbf{x}^{(k)}))) \end{aligned} \tag{2.25}$$

Quantity 2.25 is called the **cross-entropy error function (CE)**. This is because it resembles the definition for entropy, with the difference of the logarithms' multipliers being integers summing to one and not probabilities. Note that the smallest possible value the quantity inside the summation can take, for a sample, is 0, and it is achieved when the model assigns a probability value of 1 to the correct label. But there is no upper bound; the bigger the failure of a model the larger the corresponding term becomes. So, intuitively at least, the cross-entropy error function seems like a good choice for a cost function.

In addition to that, using the very useful property of the derivative of the sigmoid function shown in equation 2.23, one can easily compute its gradient and may be surprised to witness a familiar form:

$$\nabla L(\boldsymbol{\vartheta}) = \sum_{k=1}^{N} (\sigma_{\boldsymbol{\vartheta}}(\mathbf{x}^{(k)}) - y^{(k)})\mathbf{x}^{(k)} \tag{2.26}$$

This form resembles the one of the derivative of the equation 2.19, yet it is not the same as $\sigma(\mathbf{x}^{(k)}, \boldsymbol{\vartheta})$ is not a linear function of the parameters. This is not a coincidence; both these models belong to a broader family of models called **Generalized Linear Models**. This topic will not be covered in the present thesis but the reader is urged to look it up on his/her own [4, 70]. Despite their similarity, the aforementioned difference between the two methods renders $\nabla L(\boldsymbol{\vartheta}) = 0$ a nonlinear system of equations in the case of logistic regression. This is where gradient descent shows its usefulness as it can gradually reduce the error and discover a solution for the parameter vector. The update rule for the GD algorithm is:

$$\boldsymbol{\vartheta}_{i+1} = \boldsymbol{\vartheta}_i - a \sum_{k=1}^{N} (\sigma_{\boldsymbol{\vartheta}_i}(\mathbf{x}^{(k)}) - y^{(k)})\mathbf{x}^{(k)} \tag{2.27}$$

Linear regression models work well when the output is linearly dependent on the input, but a generalized linear regression model may be needed in the presence of nonlinearities. In the same way, and highlighting a certain duality between linear and logistic regression again, logistic regression, the way it was defined above, is good at separating classes that are **linearly separable**. That is, when there is a hyperplane that separates all points of the two classes, with samples of one class belonging to one side of the plane and samples of the other class belonging to the other

side. In the case of a 1-dimensional input this hyperplane is reduced to a point and in the case of a 2-dimensional input to a line as shown on the left side of figure 2.4.

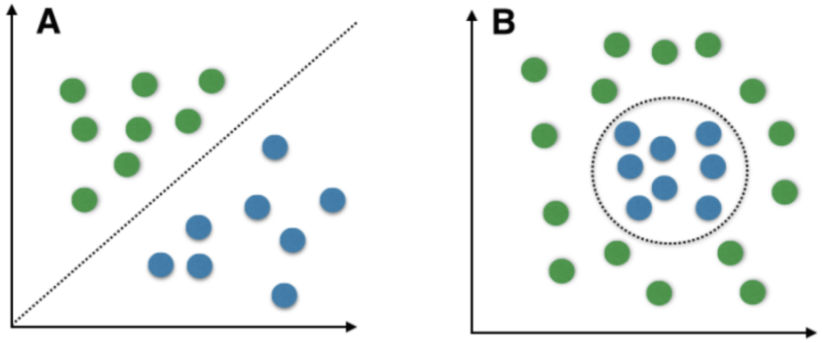


Figure 2.4. *A. Classes that are linearly separable B. Classes that aren't. Figure from* `https://www.tarekatwan.com/index.php/2017/12/methods-for-testing-linear-separability-in-python/#fn-102-2`.

But, in cases like the one shown on the right side, one has to deploy the same trick of using hand-made basis functions. Then, the update rule becomes:

$$\vartheta_{i+1} = \vartheta_i - a\sum_{k=1}^{N}(\sigma_{\vartheta_i}(\varphi(\mathbf{x}^{(k)})) - y^{(k)})\varphi(\mathbf{x}^{(k)}) \tag{2.28}$$

Multiclass Logistic Regression

Sigmoid seems like a good choice for the 2-class classification problem. But what if there are $M, M > 2$, classes that must be separated? For this problem, the **softmax function** is employed. For a class $l, l \in 1, 2, \cdots, M$, the softmax score of an input vector $\mathbf{x}$ is defined as:

$$softmax_{\vartheta,l}(\mathbf{x}) = s_{\vartheta,l}(\mathbf{x}) = \frac{exp(\vartheta_l^T\mathbf{x})}{\sum_{j=1}^{M} exp(\vartheta_j^T\mathbf{x})} \tag{2.29}$$

Obviously, the softmax scores for all classes add up to 1, which is intuitively correct. For the determination of the cost function MLE will again be used. But, instead of a Bernoulli, a multinomial distribution is now considered:

$$p(y = \omega_l|\mathbf{x};\vartheta) = softmax_{\vartheta,l}(\mathbf{x}), \forall l \in 1, 2, \cdots, M$$

Then, the negative log-likelihood is computed as:

$$\begin{aligned} L(\vartheta) &= -\ln\prod_{k=1}^{N}\prod_{j=1}^{M} p(y^{(k)} = \omega_j|\mathbf{x}^{(k)};\vartheta)^{1\{y^{(k)}=\omega_j\}} \\ &= -\sum_{k=1}^{N}\sum_{j=1}^{M} 1\{y^{(k)} = \omega_j\}\ln\, s_{\vartheta,j}(\mathbf{x}^{(k)}) \end{aligned} \tag{2.30}$$

The derivative of the softmax, that computes the softmax score for a class ω_l w.r.t. to the

parameters that correspond to the class ω_p, conveniently is:

$$\frac{\partial s_{\boldsymbol{\vartheta},l}(\mathbf{x})}{\partial \boldsymbol{\vartheta}_p} = s_{\boldsymbol{\vartheta},l}(\mathbf{x})(I_{lp} - s_{\boldsymbol{\vartheta},p}(\mathbf{x}))\mathbf{x}^T \quad \text{where } I_{lp} \text{ are the elements of the identity matrix} \tag{2.31}$$

Equation 2.30 is the cross-entropy error function for the multiclass classification problem. Again, one notices that the error term is minimized to 0 for a sample to which the model assigns a probability of 1 as to belong to the correct class. Using equation 2.31 one can compute the derivative of the cost function w.r.t. one of the parameter vectors $\boldsymbol{\vartheta}_l$:

$$\nabla_{\boldsymbol{\vartheta}_l} L(\boldsymbol{\vartheta}) = \sum_{k=1}^{N} (s_{\boldsymbol{\vartheta},l}(\mathbf{x}^{(k)}) - 1\{y^{(k)} = \omega_l\})\mathbf{x}^{(k)} \tag{2.32}$$

The same form is revealed hinting the relationship to the Generalized Linear Models family.

Again one can choose a more complicated function of the inputs as the feature vector, $\boldsymbol{\varphi}(\mathbf{x})$, in which case equation 2.32 becomes:

$$\nabla_{\boldsymbol{\vartheta}_l} L(\boldsymbol{\vartheta}) = \sum_{k=1}^{N} (s_{\boldsymbol{\vartheta},l}(\boldsymbol{\varphi}(\mathbf{x}^{(k)})) - 1\{y^{(k)} = \omega_l\})\boldsymbol{\varphi}(\mathbf{x}^{(k)}) \tag{2.33}$$

2.3.7 Other Ways of Performing Optimization

Gradient descent is a one of the simplest optimization algorithms, but advancing with knowledge only of the first-order derivatives can be very slow in some cases. This is why, many times, second-order derivatives are computed and used to speed up the process. The knowledge of both the slope and the curvature of the function can be used to determine not only the direction but also the size of the step that will be taken.

An example of an algorithm that uses second-order derivatives is **Newton's method**. What Newton's method actually does at each step is minimizing the quadratic approximation of a twice-differentiable function f around a central point $\mathbf{x}_i$ and then using the point at which the approximation takes its minimum value, $\mathbf{x}_{i+1}$, as the new starting point for the next step [8]. First, the second-order Taylor expansion of f is computed:

$$f(\mathbf{x}_i + \mathbf{e}) \approx f(\mathbf{x}_i) + \nabla^T f(\mathbf{x}_i)\mathbf{e} + \frac{1}{2}\mathbf{e}^T\mathbf{H}\mathbf{e}, \text{ where } \mathbf{H} \text{ is the Hessian of } f \text{ at } \mathbf{x}_i \tag{2.34}$$

Differentiating the right part of equation 2.34 w.r.t. $\mathbf{e}$ and finding the minimum:

$$\mathbf{e} = -\mathbf{H}^{-1}\nabla f(\mathbf{x}_i) \tag{2.35}$$

Then $\mathbf{x}_i + \mathbf{e}$ is chosen as the next starting point. Note that using a learning rate is unnecessary since all aspects of the new point are defined by $\mathbf{e}$. The resulting algorithm for cost function optimization is algorithm 2.1, where $\mathbf{H}^{-1}$ is the inverse matrix of the Hessian of $J(\boldsymbol{\vartheta})$ at $\boldsymbol{\vartheta}_i$.

ALGORITHM 2.1: *Newton's Method*

initialize $\boldsymbol{\vartheta}$ with $\boldsymbol{\vartheta}_0$
$i \leftarrow 0$
while algorithm has not converged **do**
  $\boldsymbol{\vartheta}_{i+1} = \boldsymbol{\vartheta}_i - \mathbf{H}^{-1}\nabla J(\boldsymbol{\vartheta}_i)$
  $i \leftarrow i + 1$
end while

The difference between gradient descent and Newton's method is shown in figure 2.5. Manually setting the step size, as in gradient descent, may cause delays because of the learning rate being set too small, or may lead to not finding a minimum at all and overshoot in the case of too large

learning rates.

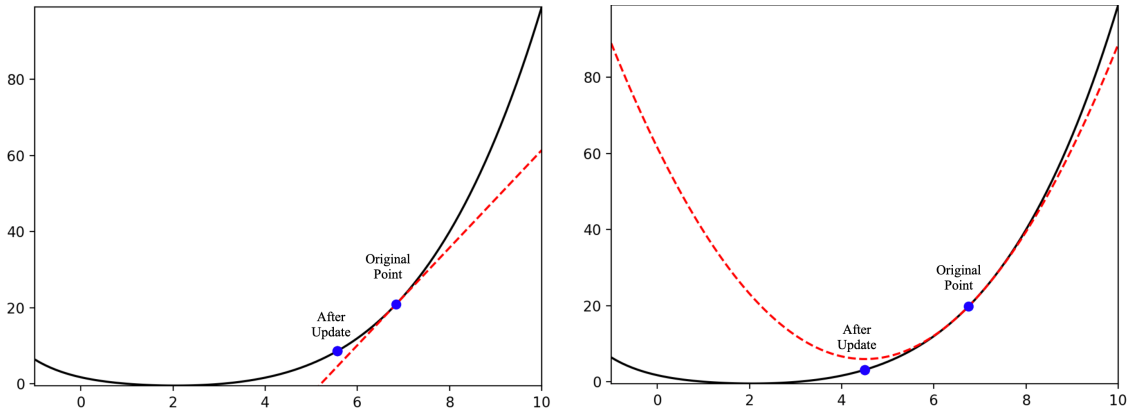


Figure 2.5. *Difference between using gradient descent (left) and Newton's method (right) for minimizing a function. Figure from* `https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote07.html`

But the smaller number of steps taken may not compensate for the larger computational burden of computing the Hessian at each step. One must then weigh the two options and, depending on the number of parameters of the model and the nature of the problem, decide which one to use [70]. Generally, second-order methods are preferred for smaller models, like GLR ones, and first-order methods for large ones, like neural networks.

2.3.8 Curse of Dimensionality

What every algorithm presented above essentially does is function approximation using data samples. For generalization purposes it is assumed that, when trying to determine the value of a newly seen sample, the values of training samples that are closer to it are more relevant than the values of training samples that lie further away, i.e. the underlying function is assumed to be smooth. Therefore, it would seem reasonable to try splitting the space into cell-shaped regions and use, for determining the value of a new sample, only data samples that exist in the same region. The problem with this approach is that the number of cells would grow exponentially with the dimensionality of feature vectors as shown in figure 2.6 or, stated in a different way, the complexity of a function grows exponentially with the input dimensionality. This fact is also referred to as the **curse of dimensionality**, a term that was introduced by Richard Bellman (1961), who was studying adaptive control at the time.

If one was to use a set of basis functions, as in the case of generalized linear regression, then, every newly added variable, that would result from an increase in the input dimensionality, would have to be combined with the existing ones in a way determined by the basis functions' definition. This aspect of the curse of dimensionality would lead to an exponential increase in the number of model parameters. Consequently, such models are impractical in the cases of high-dimensional spaces.

The good news is that real-world datasets tend to have their samples gathered near nonlinear manifolds of dimensionality much smaller than the input one. What is therefore missing, is the freedom that will allow a model to concentrate on these sub-regions instead of wasting resources paying attention to uninformative parameter combinations. The models that will be presented in chapter 2.4 are, by construction, allowed to do this and achieve linear parameter increase with respect to the input dimension [4, 8].

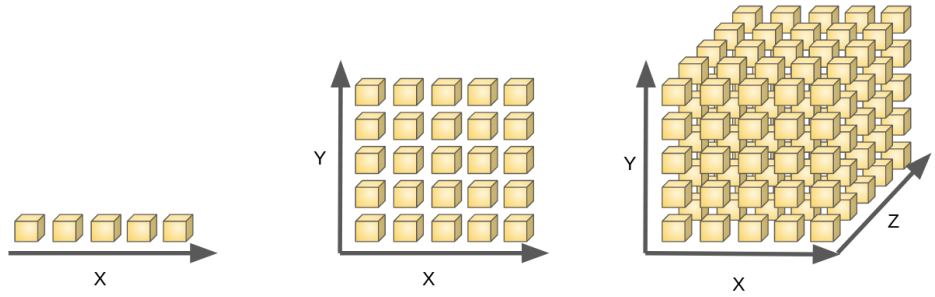


Figure 2.6. *A linear increase in the dimensionality of the input leads to an exponential growth of modeling parameters. Figure from* `https://www.i2tutorials.com/what-do-you-mean-by-curse-of-dimensionality-what-are-the-different-ways-to-deal-with-it/`

2.4 Neural Networks

It is a very interesting fact that neural networks (NNs), the basis of the most successful and influential machine learning models up to date, was an idea inspired by the human brain. The human brain is a highly complex nonlinear system, in which cells called **neurons** form clusters called **nuclei**, which then work in parallel on processing various stimuli. Neurons are local processing units, that are about six orders of magnitude slower than modern logic gates. Yet, the human brain is able to perform complex information processing tasks, such as perceptual recognition tasks, in a matter of a few hundred milliseconds with a high degree of fault tolerance. This is partly due to the parallelization of processing and the efficient way in which neurons and neural clusters are interconnected.

It is also because of the existence of an enormous number of neurons and of interconnections between them, also known as *synapses*, in the human cortex, which are thought to be near 10 billion and 60 trillion respectively. Synapses are used to transfer information between neurons. Information is received by the *dendrites* of neurons, processed by the cell's body, also called *soma*, and then transmitted by the *axons* to other neurons. Every neuron cell may communicate with thousands of others, receiving and transmitting information. A sketch of a neuron is shown in figure 2.7.

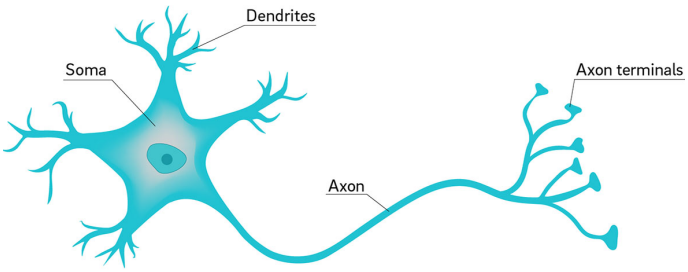


Figure 2.7. *Important parts of a neuron. Figure from* `https://today.ucsd.edu/story/why_are_neuron_axons_long_and_spindly`

The most important characteristic of the human brain is that it does not remain unchanged after the person's birth but it continues to evolve throughout the person's lifetime. It adapts to changes in a person's environment, creates specialized structures according to current needs and can even be trained at will when the owner wishes to acquire a new skill. This brain's ability is called **plasticity** and it has inspired the creation of a family of models called **neural networks** [8].

When a person learns something new, such as a new piece of information or a new skill, this knowledge is literally imprinted on the way neurons are connected in the brain. The simultaneous activation of a group of neurons causes their connections to strengthen and the transmission speed

to increase. This is either done by the strengthening the corresponding synapses or even by creating new dendrites. This process of forming long-term memories is called long-term potentiation [71].

2.4.1 Artificial Neurons

Artificial neurons are the basis of most modern neural networks. It is beneficial to consider artificial neurons as being an upgraded version of basis functions used by Generalized Linear Regression models, which were discussed in chapter 2.3.5. What is different is that these functions are now susceptible to change via a list of parameters, whose values adapt during a training procedure much like the gradient descent algorithm (algorithm 2.1) that was used to update the parameters of the linear and logistic regression models covered in chapters 2.3.5 and 2.3.6 respectively [4].

The main components of an artificial neuron are shown in figure 2.8. The input to an artificial neuron is a vector $\mathbf{x} = [x_1, x_2, \cdots, x_D]$. The input is weighted by a weight vector $\mathbf{w} = [w_1, w_2, \cdots, w_D]$ and a number called bias and denoted by b, is then added to the linear combination. In analogy to the human brain, a weight represents the strength of a synapse. But, unlike real synapses that can only suppress or amplify neural signals, the value of a weight of an artificial neuron can also be a negative number. The bias increases or decreases the input of the next module which is called an activation function, f. Activation functions are always non-linear and play a very important role in modern neural networks that will soon become apparent. The equations that describe an artificial neuron consequently are:

$$u_l = \sum_{j=1}^{D} w_j x_j \tag{2.36}$$

$$a = f(u_l + b) \tag{2.37}$$

, where the intermediate result is also called a linear combiner. Equivalently one may write:

$$u_f = \sum_{j=0}^{D} w_j x_j,\ a = f(u_f), \text{ where } x_0 = +1 \text{ and } w_0 = b \tag{2.38}$$

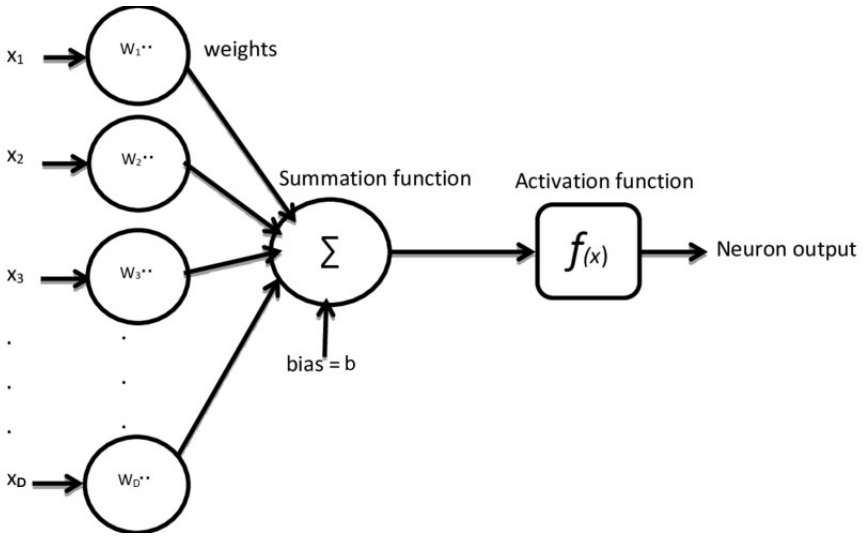


Figure 2.8. *Artificial Neuron. Figure from* `https://www.researchgate.net/publication/328733599_Impact_of_Artificial_Neural_Networks_Training_Algorithms_on_Accurate_Prediction_of_Property_Values`

The aforementioned variables are the ones that offer the possibility of plasticity, as it is in them that information learned from the training process is stored [8].

Perceptron

Before moving on to more convoluted concepts, like the Multi-Layer Perceptron, it is instructive to have a look at the training procedure of the first model that could actually be trained under a supervised learning setting, which is called **Perceptron**. The Perceptron model was created by Rosenblatt in 1958 [72]. Perceptron was based on the first artificial neuron, that was proposed by McCulloch and Pitts in 1943 [9] and its main characteristic is that its activation function is a signum function. It is used for classification tasks, assigning examples that yield positive u_f values to one class and examples that yield negative u_f values to the other class:

$$\begin{aligned} &\text{if } \mathbf{w}^T\mathbf{x} > 0 \Rightarrow \text{assign } \mathbf{x} \text{ to } \ell_1 \\ &\text{if } \mathbf{w}^T\mathbf{x} \leq 0 \Rightarrow \text{assign } \mathbf{x} \text{ to } \ell_2 \end{aligned} \tag{2.39}$$

The model is trained in an iterative process, in which the error computed on a single example is used at every iteration. This is contradictory to what was shown in the cases of linear and logistic regression that use all available training examples at each iteration to train the respective models. Let $\mathbf{x}_i$ denote the example used at the i-th iteration and $\mathbf{w}_i$ denote the version of the weight vector before the i-th update. Then the model's parameters are updated as is shown below:

Algorithm 2.2: *Perceptron's Training Algorithm*

initialize $\mathbf{w}$ with $\mathbf{w}_1$
$i \leftarrow 1$
while $\exists$ example not correctly classified **do**
  if $\mathbf{w}_i^T\mathbf{x} \leq 0$ and $\mathbf{x} \in \ell_1$ **then**
    $\mathbf{w}_{i+1} = \mathbf{w}_i + \eta_i\mathbf{x}_i$
  else if $\mathbf{w}_i^T\mathbf{x} > 0$ and $\mathbf{x} \in \ell_2$ **then**
    $\mathbf{w}_{i+1} = \mathbf{w}_i - \eta_i\mathbf{x}_i$
  else $\mathbf{w}_{i+1} = \mathbf{w}_i$ ▷ If $\mathbf{x}_i$ is correctly classified do not change $\mathbf{w}$
  end if
  $i \leftarrow i + 1$
end while

Rosenblatt also proved what is known as the **Perceptron Convergence Theorem**.

Θεώρημα 2.1. *If the two classes are linearly separable, then algorithm 2.3 will converge to a correct solution within a finite number of steps.*

In essence, the algorithm shifts the separating hyperplane towards the direction needed to classify correctly the wrongly classified example. Nevertheless, it is possible for the model to continue failing at the classification of the example even after the shift, if the learning rate of the iteration η, which is defined by the user, is not big enough. But, by repeating the said process it is supported by the theorem 2.1 that the hyperplane will eventually shift to a correct position [9].

2.4.2 Feed-Forward Neural Networks

As stated in chapter 2.3.8, models like linear and logistic regression lack the flexibility needed to model functions that become highly complex only inside small sub-regions of the input space but remain relatively simple outside of them. Artificial neurons' parameters, like the Perceptron's ones, can change through training processes, but artificial neurons can only model very simple functions. It then comes natural to try using artificial neurons as building blocks of models that will hopefully provide the desired flexibility. These models are called **neural networks**.

The simplest architecture one can build by combining multiple artificial neurons is a single-layer neural network, in which the inputs are fed to artificial neurons. Their outputs are then linearly combined, much like in Generalized Linear Regression models. The term single-layer refers to the

output layer of computational nodes as the input or source nodes do not perform any computation.

By stacking a number of such layers one on top of the other and using the outputs of one as inputs to the next a **multi-layer neural network** is created. The layers between the source and the output node are called **hidden layers**, as they are neither seen by the input nor by the output. This architecture enables the extraction of higher-order statistics of the input, as the existence of more than one layers of processed information allows the net to acquire a global perspective, despite its units being locally connected [73].

A 4-layer neural network is shown in figure 2.9. This type of neural network is also called a **feed-forward neural network** (FFNN), as the computational flow does never create a circle or, in other words, there are no recursions.

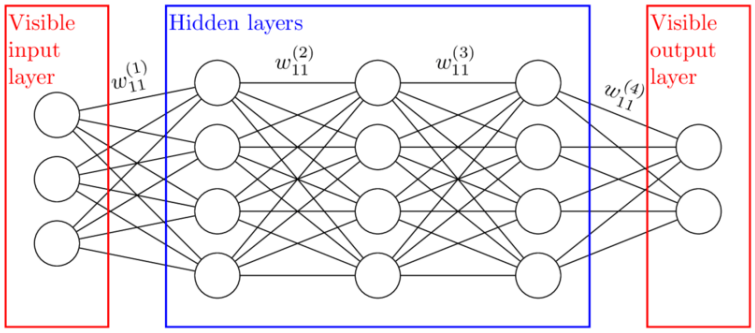


Figure 2.9. *A 4-layer Feed-Forward Neural Network, with 4 neurons in each of the hidden layers and 2 neurons in the output layer. Figure from* `https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning`

The equations of a multi-layer feed-forward network result naturally from the equations of the artificial neuron, 2.38, and the rule that the inputs to a layer are the outputs of the previous layer. A ffnn of L layers has $L-1$ hidden layers which are followed by an output layer and the l-th layer consists of D_l neurons. Moreover, the weight that multiplies the output of the i-th neuron of the l-th layer, $a_i^{(l)}$, which is then used as input to the j-th neuron of the $(l+1)$-th layer is denoted by $w_{ij}^{(l+1)}$. Finally an input of dimension D_{in} and an output of dimension D_{out} are assumed. Then the output of the i-th neuron of the first hidden layer is equal to:

$$a_i^{(1)} = h(\sum_{d=0}^{D_{in}} w_{id}^{(1)} x_d) \tag{2.40}$$

where $\mathbf{x} = [x_1, x_2, \cdots, x_{Din}]^T$ is the input to the ffnn, $x_0 = +1, w_{i0}^{(1)} = b_i^{(1)}$ is the bias added to the weighted sum of the respective neuron and h is the activation function applied. In the same spirit, the output of the j-th neuron of the l-th layer is equal to:

$$a_j^{(l)} = h(\sum_{d=0}^{D_l} w_{jd}^{(l)} a^{(l-1)}) \tag{2.41}$$

The number of hidden layers, of neurons at each layer and the type of activation functions are hyper-parameters whose values are usually chosen after an extensive search that will be discussed later in this chapter. What is implicitly assumed in the above equation is that the activation function is the same for all neurons of the network. Yet, sometimes the neurons of the output layer use a different activation function that depends on the type of the label. Usually no activation function is used in regression problems, a sigmoid function is used for binary classification problems and a softmax function is used in the case of K-class classification problem [4].

Global Approximation Theorem

To comprehend the expressive strength of ffnns it suffices to take a look at the theorem known as **Global Approximation Theorem** [74, 75].

Θεώρημα 2.1. *A two-layer feed-forward neural network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided that the net has a sufficiently large number of hidden units.*

This theorem holds for a variety of activation function types. Nonetheless the theorem does not explain how to find the optimal model hyper-parameter and parameter values for a certain task and dataset. It is only indicative of the potential power of ffnns. In the following chapter, the second problem will be discussed, in the context of ffnns.

2.4.3 Backpropagation Algorithm

In chapters 2.3.5 and 2.3.6 it was shown how the problem of training a linear and a logistic regression model can be approached from a probabilistic perspective. This was done by assuming that labels follow certain probability distributions and then using MLE to find the cost function which can by minimized by using an optimization algorithm like GD. This was easily extended to the case of models that use unalterable nonlinear basis functions, equations 2.19 and 2.28.

The goal is to further extend this method to the case of basis functions with trainable parameters. Training a neural network means training the weights of all artificial neurons it is made of so that a cost function $J(\mathbf{w})$ is minimized. The problem is that, while the contribution of each trainable parameter of a GLR model is independent of the contribution of the others, the same is not true in a ffnn. In a ffnn the value of a parameter, which shapes a neural output, is then propagated through the rest of the net and therefore interacts with parameters of other neurons. This poses a problem to the calculation of the gradient values for these parameters. An algorithm that tackles this problem and is currently used as part of the training process of most modern neural networks is the **Backpropagation algorithm**. The gradients it computes can be used for example in the well known gradient descent updates, in which the weight vector $\mathbf{w}$ is corrected by $\Delta\mathbf{w}$, where $\Delta\mathbf{w}$ is computed as to point to the direction of the greatest rate of decrease of $J(\mathbf{w})$, $-\nabla_{\mathbf{w}} J(\mathbf{w})$.

Starting with the gradients w.r.t. weights that belong to output neurons, computing them is quite straight-forward. One has to simply observe that equations 2.19, 2.28 and 2.33 also apply here since the rest of the net plays the role of the basis functions. But to compute gradients w.r.t. the rest of the weights, the equations of a ffnn, 2.40 and 2.41, and the chain rule of differentiation have to be deployed. Because $J(\mathbf{w}) = \sum_{k=1}^{N} J_k(\mathbf{w})$, where N is the number of training samples, $J_k(\mathbf{w})$ will be used from now on to simplify the notations. Using this simplification, the equations used for an output weight $w_{ji}^{(L)}$ can be expressed as:

$$\frac{\partial J_k}{\partial w_{ji}^{(L)}} = \frac{\partial J_k}{\partial u_j^{(L)}} \frac{\partial u_j^{(L)}}{\partial w_{ji}^{(L)}}, \text{ where } u_j^{(L)} \text{ is summed input to the } j\text{-th neuron of the final layer} \quad (2.42)$$

Then, the notation

$$\delta_j^{(L)} = \frac{\partial J_k}{\partial u_j^{(L)}} \quad (2.43)$$

is introduced. It is also true that

$$a_i^{(L-1)} = \frac{\partial u_j^{(L)}}{\partial w_{ji}^{(L)}} \quad (2.44)$$

where $a_i^{(L-1)}$ is the output of the i-th neuron of the $(L-1)$-th layer. By applying equations 2.43 and 2.44 to equation 2.42 the latter is written as:

$$\frac{\partial J_k}{\partial w_{ji}^{(L)}} = a_i^{(L-1)} \delta_j^{(L)} \tag{2.45}$$

But, when considering the derivative w.r.t. a weight of a random neuron $n_i^{(l)}$, that is the i-th neuron of the l-th hidden layer, with output $a_i^{(l)}$, one has to observe a dependence that will be now discussed to be able to perform the computations efficiently. Using the chain rule one arrives to the conclusion that:

$$\delta_i^{(l)} = \frac{\partial J_k}{\partial u_i^{(l)}} = \sum_{d=1}^{D_{l+1}} \frac{\partial J_k}{\partial u_d^{(l+1)}} \frac{\partial u_d^{(l+1)}}{\partial u_i^{(l)}} \tag{2.46}$$

Essentially, the effect of a change at $u_i^{(l)}$ can be summed by considering the effect of each neuron that comes directly after $n_i^{(l)}$ in the computational path multiplied by their local interaction. Note that if some neuron $n_j^{(l+1)}$ is not connected to $n_i^{(l)}$ then the corresponding local interaction is equal to zero and this path is not added to the sum. Using the following definition for δ:

$$\delta_i^{(l)} = \frac{\partial J_k}{\partial u_i^{(l)}} \tag{2.47}$$

and considering that:

$$\frac{\partial u_j^{(l+1)}}{\partial u_i^{(l)}} = w_{ji}^{(l+1)} h'(u_i^{(l)}), \text{ where } h \text{ is the activation function} \tag{2.48}$$

one can write the equation 2.46 as:

$$\delta_i^{(l)} = h'(u_i^{(l)}) \sum_{d=1}^{D_{l+1}} \delta_d^{(l+1)} w_{dl}^{(l+1)} \tag{2.49}$$

Equation 2.49 shines light on a dependence between values of δ of neurons of consecutive layers and means that once these values are computed for a certain layer $l+1$, then they can also be computed for the previous layer in time linear to the number of weights of the latter layer.

Moreover, similarly to equations 2.43 and 2.44, it can be written about the corresponding quantities regarding previous layers that:

$$\frac{\partial u_j^{(l)}}{\partial w_{ji}^{(l)}} = a_i^{(l-1)} \tag{2.50}$$

$$\frac{\partial J_k}{\partial w_{ji}^{(l)}} = a_i^{(l-1)} \delta_j^{(l)} \tag{2.51}$$

Using equations 2.49, 2.50 and 2.51 the problem of efficiently computing the gradients of a ffnn is finally solved and what is left is placing all the pieces together. The Backpropagation algorithm consists of two main phases. The first one is the computation of the activation functions of all of its neurons. For ffnns, this means starting from the input and iteratively computing the output of every layer that follows until the final layer using the ffnns' equations, 2.40 and 2.41.This step is referred to as **forward propagation**. The second step is the computation of the δ's starting from the output units using 2.43, 2.44 and 2.45 and continuing backwards using equations 2.49, 2.50 and 2.51. This recursive process is referred to as **backward propagation**. This can be formulated in

the algorithm shown following algorithm.

ALGORITHM 2.3: *The Backpropagation Algorithm*

Apply an input and forward propagate through the neural network
Using the label of the input compute the error
Compute $\delta_j^{(L)}$
Backpropagate the δ's
Use the δ's and the activation values to compute the gradients

Since both the forward and the backward propagation cost $O(dim(\mathbf{w}))$ the overall cost is equal to $O(dim(\mathbf{w}))$, which is linear w.r.t. to the number of weight of the network. It is also important to note that the backpropagation algorithm can be used in many kinds of neural networks and not just ffnn and with many types of error functions.

2.4.4 Notes on Training Neural Networks

As was noted in the previous chapter, the Backpropagation algorithm is used to compute the gradients of the error w.r.t. the weights of a neural network. This can be a part of a gradient descent process. There are a few matters that ought to be discussed before concluding the matter of training neural networks.

Learning Curves

When training a machine learning model with an iterative process it is useful to be able to know if and how fast the model is getting better through this process. This can be seen by observing the evolution of the error computed on the training set while the training process proceeds. Because the weights are usually randomly initialized, the error on the training set is initially large and gradually becomes smaller as the model is trained. After some training iterations, the error reaches an asymptotic value that depends on the Bayes error, the quality and quantity of the data and the expressive capabilities of the model, as was discussed in chapters 2.3.1 and 2.3.3. So far only the case of using, for each training step, the gradients computed using the entire training set has been discussed. But, as will be explained shortly, sometimes only a part of the training set may be used for each update. In these cases, the training set is split into B non-overlapping subsets of samples, called **batches**, and for each weight update the gradients computed using the error on samples of a single subset are used. After the algorithm has performed B updates using a different batch each time, it splits the training set again and repeats the same process. Each of these cycles consisting of B updates is called an **epoch**. Obviously, when the entire training set is used for an update it is true that $B = 1$ and the epoch only consists of one update.

Apart from the error on the training set it is useful to continuously monitor the performance of the model, that is being trained, on samples that it hasn't been trained on, or as researchers usually say, on samples that it has never seen. This is a good test, as to whether the model is acquiring generalizing capabilities or is simply remembering the input - output map-

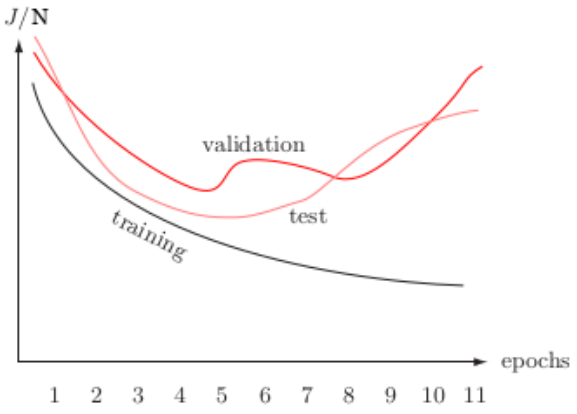


Figure 2.10. *Learning curves for a 4-layer ffnn. The normalized error is used, $J/N = 1/N \sum_{k=1}^{N} J_k$ [3].*

pings of the training examples. The set of the samples that are used for this purpose is called a **validation set**. As shown in figure 2.10 and can be easily understood, the errors on the validation and the test set are usually higher than the one on the training set [3].

Error Surfaces

Since training is actually an optimization process it is sometimes useful when designing a training algorithm to view it from the perspective of a descent of the error surface. Specifically, each step taken in the direction of $\delta\mathbf{w}$ changes J by by $\delta J \simeq \delta\mathbf{w}^T \nabla J(\mathbf{w})$. As noted in the previous chapter $\delta\mathbf{w}$ is usually chosen to point to the direction of the maximum rate of decrease of J. In contrast to the Perceptron, modern neural networks generally use differentiable activation functions that render J a smooth continuous function of $\mathbf{w}$. Therefore, at the global minimum of J it is true that $\nabla J = 0$.

There are multiple problems one might encounter during the training process that are related to the error surface of the problem. Most of them are caused by the complexity of the neural models, the outputs of which result from highly nonlinear combinations of their parameters. One of the most commonly observed ones is related to the existence of search sub-spaces in which the error surface is relatively flat, leading the resulting gradients inside these sub-spaces to be small. This results to slow rates of convergence that can be discovered by the researcher thanks to a stagnancy of the learning curves. The opposite form of difficulty results from overly steep error surfaces that lead that lead to big gradient values which may cause the algorithm to completely miss the minimum by jumping over it. This problem is known as **overshooting**.

The exploration of the global minimum is also hindered by the existence of multiple **local minima**. A training algorithm may become stuck in a local minimum $\mathbf{w}^*$ since reaching the global minimum from $\mathbf{w}^*$ may require the descending algorithm to explore, for a few iterations, regions with larger errors than the one corresponding to the local minimum. Interestingly though, it has been observed that local minima many times yield errors that are close to global ones, and consequently constitute valid solutions to the training problem[insert research paper] [3, 4, 8].

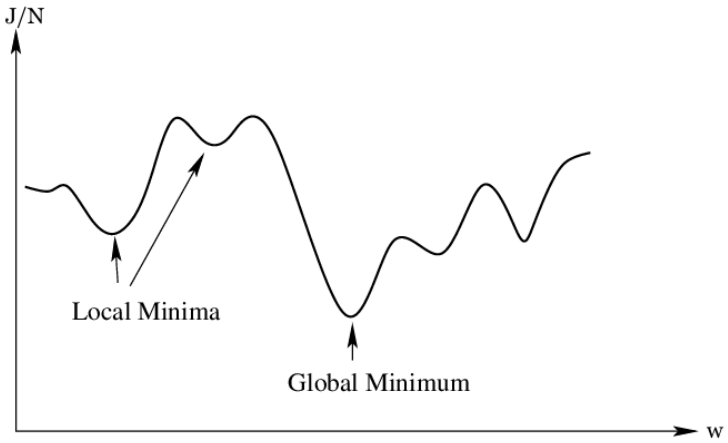


Figure 2.11. *Error Surface of a model with 1 parameter whose value spans on the horizontal axis. Figure from* `https://inverseai.com/blog/gradient-descent-in-machine-learning`

A lot of research effort has been devoted to discovering training methods that tackle the problems presented above, especially the last one. Some of them will be discussed later in this chapter.

Training Protocols

The GD algorithm, as was described in chapter 2.3.5, computes the error on the entirety of the training set, proceeds with calculating the gradients for each training instance and then combines them to perform the update. This method provides an accurate estimation of the updating vector,

$\delta\mathbf{w}$, and, for reasonable learning rate values, guarantees the convergence of the training process to a local minimum. Moreover, since the computations performed for each training example are independent of one another, the parallelization of this procedure is possible and accommodated by modern hardware accelerators.

Nevertheless, it is sometimes preferable not to use the full training set when performing an update, but only a part of it. One can even use the error computed on a single training example to update the parameters of the model. In this case the algorithm randomly chooses and uses one sample at every iteration until all training samples have been used, at which point it is said that an epoch has been completed. This process is named **stochastic training** due to the inherent randomness in it. One of the biggest advantages of this method is its minimal storage requirements in contrast to first one, for which the activation signals and gradients for all examples have to be stored during the parallelization process. In addition to that, stochastic training is good at dealing with redundancies as it avoids repeating unnecessary computations in a single update. Finally, observe that the algorithm descends a different error surface at each update because of the uniqueness of the error function. Therefore, since it is highly unlikely that a local minimum will coexist in many error surfaces, the algorithm will be able to escape many local minima using updates, the error surfaces of which, will not have local minima in the same spots.

Unfortunately, stochastic training is an easily parallelizable process and it also tends to produce noisy learning curves because descending the error surface of a training sample does not ensure the descent of the error surface corresponding to the total error. In order to tackle these problems, researchers usually compute the error on a number B, $1 < B < N$, of training samples simultaneously. B is referred to as batch size. This method seeks and achieves the best of both worlds and today is utilized by most ML training algorithms [3, 4, 8].

Stopping Criteria

The issue of the choice of the stopping point of the training process has not been addressed yet. Multiple stopping criteria have been proposed in the literature, as to when to stop training the model weights.

A simple criterion results from observing that the algorithm must stop nearby a global or a local maximum. Since the error surface is smooth, this means that gradient magnitudes will be relatively small close to these points. A reasonable criterion would thus be to consider the algorithm to have converged when the Euclidean norm of the gradient vector has reached a user-defined relatively small threshold. The problem with this approach is that, in the case of stochastic training, it may take long for the algorithm to reach such a spot.

Similarly, one can use the change of the cost function's magnitude as a metric for convergence. Specifically, assume convergence when the absolute rate of change in the average error per epoch is small enough. A possible disadvantage is the existence of the possibility of premature termination since the value of the threshold is almost never known beforehand.

A theoretically supported approach is the use of the error on a set of samples that the model has never seen as a convergence indicator. This set can be the validation one. Information of this curve is indicative of the model's generalizing capabilities and therefore a valid stopping criterion [3, 8].

Vanishing Gradient, Activation Functions and Residual Connections

Before discussing the issue of choosing an activation function for the hidden layers, it is necessary to have a look at one of the most well-known difficulties when it comes to neural network training. As suggested by the Global Approximation Theorem, 2.1, a two-layer ffnn can approximate any

continuous function, given a sufficient number of hidden units. Yet, the recent success of ML has been heavily supported by the use of neural networks with multiple layers, otherwise known as **deep nets**. The reasons as to why this happened will be discussed later in this chapter. For now, the main focus will be on the difficulties of training such nets, and also on the way that a certain type of activation functions and a structural trick allowed researchers to overcome them.

In order for a parameter to be trained, the gradient of the error w.r.t. to this parameter must first be computed. This is then used to shift the parameter towards the direction of the greatest decrease in the error function. The problem with standard deep neural networks is that the gradients w.r.t. parameters that belong to neurons of early layers tend to become increasingly small as the net becomes deeper. As a result, the training speed of these parameters is considerably slower than the one of parameters of neurons that are closer to the output of the network. This problem is referred to as the **vanishing gradients** problem and it was first introduced in the context of recurrent neural networks (RNNs) [76], that will be introduced in chapter 2.6. To understand why this is the case, it is appropriate to view the following example of a 3-layer neural network with one neuron per layer [77] shown in figure 2.14.

The update rule for the first weight of this network is the following: $\mathbf{w}_{i+1}^{(1)} = \mathbf{w}_{i}^{(1)} - \eta \nabla_{\mathbf{w}} J$.

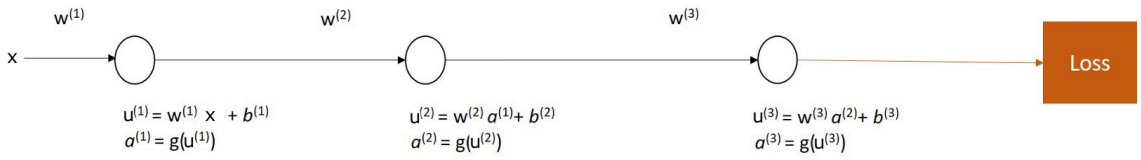


Figure 2.12. *3-layer ffnn with one neuron per layer. Figure from* ***https://towardsdatascience.com/vanishing-gradient-in-deep-neural-network-83953217c59f***

In the case of a 1-dimensional weight, and assuming all biases to be equal to zero, it follows that:

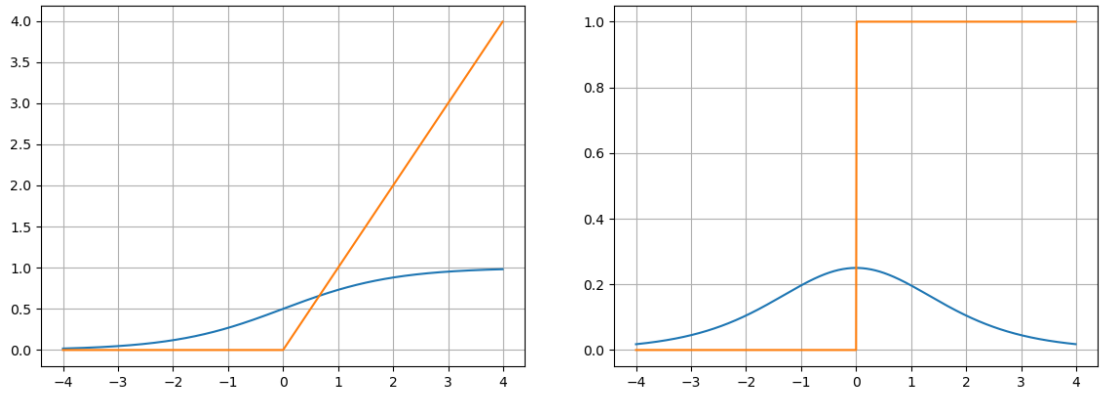
$$\begin{aligned} \frac{\partial J}{\partial w^{(1)}} &= \frac{\partial J}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial u^{(3)}} \frac{\partial u^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial u^{(2)}} \frac{\partial u^{(2)}}{\partial a^{(1)}} \frac{\partial a^{(1)}}{\partial u^{(1)}} \frac{\partial u^{(1)}}{\partial w^{(1)}} \\ &= \frac{\partial J}{\partial a^{(3)}} \frac{\partial u^{(3)}}{\partial a^{(2)}} \frac{\partial u^{(2)}}{\partial a^{(1)}} \frac{\partial u^{(1)}}{\partial w^{(1)}} \prod_{l=1}^{3} \frac{\partial a^{(l)}}{\partial u^{(l)}} \end{aligned} \tag{2.52}$$

The vanishing gradient problem may appear in two main cases:

- if at least one of the derivatives related to the activation function, $\frac{\partial a^{(l)}}{\partial u^{(l)}}, l \in \{1, 2, 3\}$ is zero then $\frac{\partial J}{\partial w^{(1)}} = 0$
- if $\frac{\partial a^{(l)}}{\partial u^{(l)}} \simeq 0$, for most of the layers $l \in \{1, 2, 3\}$ then $\frac{\partial J}{\partial w^{(1)}} \simeq 0$

A widely used activation function that creates the vanishing gradients problem is the sigmoid function, 2.22. The sigmoid function has several desired properties. It is continuous and smooth, meaning that its derivative exists, and even better, it can be easily computed using the equation $\sigma'(x) = \sigma(x)(1 - \sigma(x))$. It is also a monotonic function, which prevents the introduction of additional local extrema. Moreover, it has a limited output range, i.e. it saturates. This helps in keeping the weights and the activation function outputs bounding, ensuring the training time to be limited. Unfortunately, this also affects the quality of training. To see why, observe the derivative of the function painted in blue in figure 2.13b [3, 4, 8].

The derivative of the sigmoid only takes relatively large values for inputs belonging to a narrow interval centered at 0, also known as the **significance region**. For inputs outside this interval the magnitude of the derivative takes much smaller values, that when are multiplied together as



(a) *Functions* **(b)** *Derivatives*

Figure 2.13. *Showing* ***a)*** *the sigmoid (blue) and the ReLU (orange) functions and* ***b)*** *their derivatives [5]*

a part of a gradient computation process, like the one described by the equation 2.52, lead to the vanishing gradient problem.

An alternative to the sigmoid function that has been proposed in the literature is the **rectified linear unit (ReLU)** [78], which is defined as follows:

$$ReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.53}$$

A graphical representation of this function is shown in figure 2.13a and its derivative is shown in figure 2.13b. The fact that its derivative for all positive values is constant and equal to 1 solves the vanishing gradient problem for these values by allowing the gradients' passage to previous layers. Of course, as a result of this, this function does not saturate and in the case of negative inputs the gradient becomes equal to zero, which means that the problem still persists in these cases. Multiple modifications to the ReLU function have been suggested to alleviate the last weakness, like the leaky ReLU [79]. This problem is also closely connected to the weight initialization one. If the network's weights are initialized in a way that gradients are close to become or are zero for multiple layers then the training of the network turns out to be extremely slow. Details about weight initialization will follow shortly.

For brevity reasons the rest of this note will simply focus on another important trick that was proposed in 2015 [6] and firstly applied to convolutional networks (CNNs) that will be presented in chapter 2.5. The trick was the addition of **residual connections**, as shown in figure, that bypass neural layers. These lines directly connect the input to the output by adding the first one to the output of the neural layer. This method allows the gradients to flow uninterruptedly and was a substantial breakthrough that contributed to the birth of very deep neural nets [4].

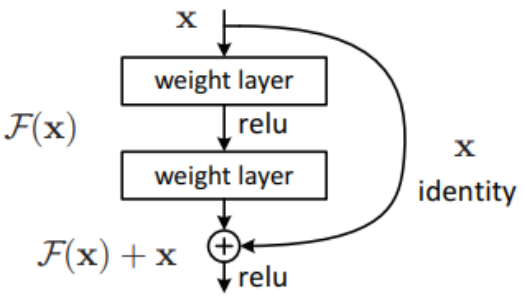


Figure 2.14. *Residual connections [6]*

Input Scaling

Clearly, NN training is extremely sensitive to the value intervals of the signals involved in the process, starting from the input ones. If, for example, the values of two signals differ by orders of magnitude, the bigger one affects the total error much more than the other. This causes the training process to focus mainly on updating its corresponding weights and, as a result, the new will pay much more attention to the respective feature than the other, even though the latter might be of equal significance when it comes to the prediction task [3].

In another example, all features take only positive values. This means that, during h=the updates, weights can only increase together or decrease together. So, in order for the weight vector to reach a specific point, it has to zig-zag through the search space slowing the training process down [8].

These problem have lead to the adoption of various data preprocessing techniques by the researchers. A widely used one is feature normalization. It entails shifting the features of all samples so that they have a zero mean and scaling them until they share the same variance, which is usually equal to 1. One may also choose a min-max approach that scales the features of the dataset so that the two extreme values that each feature takes become equal to a predetermined min and max value respectively. As a part of the data preprocessing procedure, and in order speed up learning, researchers also use to create uncorrelated input variables using a numerical transformation technique like PCA [8, 80].

Note that data preprocessing happens only once before the training begins, so it is an overhead that is paid only once.

Target Values

When it comes to target values it is required that they must be set within the range of the output activation function h. In fact, they are often offset by ϵ away from the endpoints of the value domain $(-a, a)$ of h: $a - \epsilon)$ and $-a + \epsilon$. Otherwise, the weights are driven to infinity by the training process and it is in turn slowed down [8].

Weight Initialization

Weights must be initialized carefully because failing to assign proper initial values to them any render a neural network, that would otherwise be suitable for the problem at hand, untrainable. If weight values are too large, neurons will be driven into saturation and local gradients will be small. If, on the other hand, they are initialized with very small values training will occur, in the case of sigmoid activation functions, on a flat area around the origin of the error surface. Thus, in both cases learning will be slow or may not even happen. As a result, the weights should be set so that the standard deviation (std) of the induced local field of the neurons lie in the transition are between the linear and the saturated intervals of its activation function. Moreover, since the input will already be normalized and there will be an approximately equal number of positive and negative values, the same must happen to the weights [3, 8].

A common way to initialize the weights is by sampling them from a uniform distribution $U_{[-1/\sqrt{d}],1/\sqrt{d}]}$, where d is the number of input to the respective neuron. Of course, many initialization techniques have been proposed and the it continues to be considered an open problem. The interested reader may find additional information in [58] and [81].

Learning Rate

Learning rate is one of the most important hyper-parameters when it comes to NN training. At first glance, a reasonable approach would be setting it to a relatively small value guaranteeing

model convergence to a local minimum. However, apart from the obvious disadvantage of delaying training, this approach would also also cause another problem. Ideally, all neurons of a NN should learn at the same speed as models that have some of their parameters trained far quicker than others are known to perform better for subsets of the training data and very bad for the rest. This is a serious problem, especially since neurons of the final layers are usually trained much faster than neurons that are closer to the input. The same has been observed to occur happen in the case of neurons with many inputs, whose parameters converge before the parameters of neurons with small input dimension. The existence of different training speeds in standard neural networks along with the need of simultaneous training of all parameters render the use of different learning rates for different neurons necessary. Even worse, as the training process evolves the form of the error surface may change and the learning rates will also have to adapt to the new training conditions [3, 8].

Note that in chapter 2.3.7 the learning rate was determined with the help of the second-order derivatives, but that is not possible in the case of NNs with several million parameters. Nevertheless, after decades of experimentation researchers have concluded to some reasonable values for learning rates, that depend on the type and the size of the model, as long as methods of changing the learning rate depending on the phase of the training process. Information regarding this issue will be provided in the next paragraph.

Momentum

Modern neural networks almost never use the gradient value of the point under examination alone when performing an update. Error surfaces sometimes contain sub-spaces where the slope is minimal, i.e. gradient's magnitude is small, called plateaus. This leads to heavy training delays. In other cases a model may find itself stuck prematurely to a local minimum delaying the training process. To alleviate these problems a modification to the standard update rule is usually applied, which is called **momentum**. In physics, momentum is a property of moving objects that tend to keep their kinetic state unless acted upon by outside forces. In the context of GD, momentum means that that the previously computed gradients will also be taken into consideration, along with the current one, when updating the parameter vector. Let w_i be a weight of a neuron in the $(i-1)$-th update. Then δw_i that is added to update the weight is defined as:

$$\Delta w_i = a \cdot \frac{\partial J}{\partial w_i} + m \cdot \Delta w_{i-1}, \text{ where } m \text{ is the momentum factor} \tag{2.54}$$

Note that, through Δw_{i-1}, all previous updates are taken into consideration. The updating vector of an update that took place k steps before the current one is multiplied with a^k. Therefore, the hyper-parameter m is usually chosen to belong to $(0, 1)$ for stability reasons, and the contribution of each vector decreases exponentially as i increases.

With the use of momentum, a parameter vector that has "gained speed" through several updates that have caused its continuous displacement cam easily quickly a plateau even if it causes gradient values to be relatively small. In the same manner the parameter vector may overcome a shallow local minimum even if the gradient orders it differently. There is an additional advantage to using momentum. Momentum tends to average out conflicting updates assisting in creating a smoother learning trajectory in error surfaces that would otherwise cause the gradient to continuously change its direction. This is also helpful in the case of stochastic learning where consecutive updates use different samples and the resulting gradients may point to entirely different directions.

Most modern optimizers use some form of momentum. For more information one the matter one can read the following papers [82, 83].

2.4.5 Generalization

It is important to remember that the main goal of the training of a neural model is not to fit the training set but to render the model capable of generalizing to examples that it has not previously seen. From a mathematical perspective, training a model is a curve-fitting problem, where the training data are the samples of the function being approximated and the network is the nonlinear input-output mapping. In that sense, generalization means that the model performs a successful nonlinear interpolation of the training samples.

A weakness that NNs are known to suffer from is overfitting, which was discussed in chapter 2.3.5. In essence what happens is that the net begins memorizing the training data and fails at fitting the underlying function the same way it described in the respective chapter. A question that has been left unanswered concerns the desired properties of the curved that is being implemented by the NN. In other words, what makes a successful interpolation? An assumption that is usually made is that the function that will be chosen must be the simplest one that fully explains the data at hand. Criteria that are imposed to ensure the model satisfies this assumption are known as **Occam's Razor** [84]. Regularization, as was described, does exactly that, eliminating unecessary parameters and thus decreasing the model's complexity. This is visually translated as an increased function's smoothness, figure 2.2.

The next question that needs answering is finding the factors that determine the generalizing abilities of the model. It is known that generalization is determined by:

- the size of the training sample
- the neural model that has been chosen
- the complexity of the underlying function that must be modeled

Starting from the last factor it is obvious that the more complex a function is the harder it will be for the NN to interpolate successfully between the training samples. This is an inherent property of the problem and, in contrast to the first two factors, can't be influenced by the researcher. Moving on to the training size, the more samples one possesses the less freedom is provided to the model to assume values for intermediate points. An empirical rule for choosing the training size is based on the Widrow's rule of thumb for the LMS algorithm [85] is the following:

$$N = O(\frac{W}{\epsilon}) \tag{2.55}$$

where N is the size of the training set, W is the number of trainable parameters of the model and ϵ is the maximum classification error allowed in the test set. For example, for an error of up to five percent, this rule determines that the number of training examples must be twenty times the number of trainable parameters of the net. Note that the number of training examples needed is directly proportionate to the number of trainable parameters, which is indicative of the expressive power of the net. This means a powerful model is more prone to overfitting than a less powerful one.

The matter of choosing and training a NN that generalizes well will be addressed next.

Bias-Variance Trade-off

In 1992, Barron [86] proved a bound to the average error on the test set of a ffnn with a hidden layer of D_1 neurons:

$$E_{av}(N) = O(\frac{C_f^2}{D_1}) + O(\frac{D_{in}D_1}{N}\log N) \tag{2.56}$$

where C_f is the first absolute moment of the Fourier magnitude distribution of the function f.

Observe that D_{in} appears in both terms. The first requirement is known as *Accuracy of Best Approximation* and becomes smaller as D_1 increases, which is also suggested by the Universal Approximation Theorem (UAT), (2.1). As D_1 gets bigger the net's expressive power is also increased. The second one is called *Accuracy of Empirical Fit to the Approximation* and in order for it to be small, D_1/N must also be small. This is directly connected to the problem of overfitting addressed in chapter 2.3.5. The combination of these terms reveals a trade-off between the expressive power of the net and its tendency to overfit. This tendency is also documented as the **bias-variance trade-off**, a designation that indicates the need of finding a balance between these two. This is pictured in the figure 2.15.

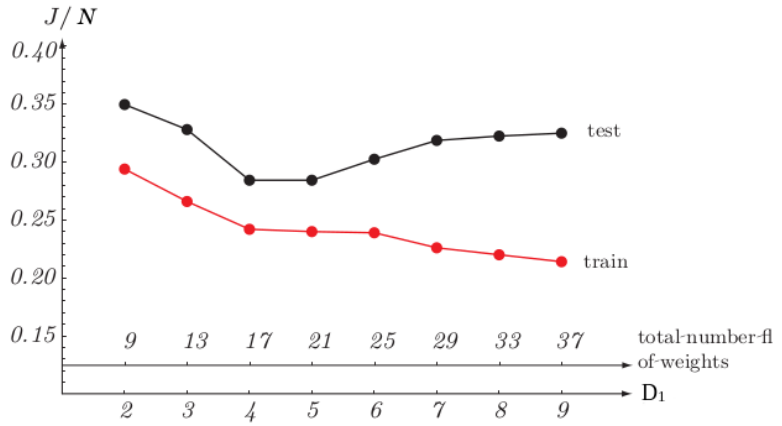


Figure 2.15. *The training and test error on a classification problem with two classes w.r.t. the number of hidden units of a 2-D_1-1 NN [3]*

For small numbers of neurons in the hidden layer the network lacks the expressive power needed to perform the task. For bigger values it can fit the training set but overfits on the test set.

What is also interesting to observe in equation 2.56 is that, in the case of NNs, an exponentially large sample size w.r.t. the input dimension D_{in} is not required to decrease the average error. The same is not true however for other types of smooth functions like polynomials and trigonometric functions. If s is the number of continuous derivatives of a function of interest, and thus a measure of the function's smoothness, then for these types of functions the minimax rate of convergence of the total risk $E_a v(N)$ is of order $(1/N)^{2s/(2s+D_{in})}$. This is a mathematical way to express the well known curse of dimensionality. The enormous advantage of NNs now becomes apparent.

Returning to the UAT, this theorem is also indicative of the expressive power of NNs with just one hidden layer. Yet, it does not state that single-layer networks are optimum in every case nor does it suggest a way to determine the optimum number of layers for a neural net. In fact, state-of-the-art NNs have multiple layers which are trained with the help of tricks analyzed in the previous chapter and a few more, like dropout and batch normalization that will be discussed in chapter 2.4.5. It is generally believed that the weakness of single-layer NNs stems from the global interaction of neurons in single-layer networks. Because of this it is difficult to improve the error that corresponds to a set of examples without increasing it for another one.

Funahashi (1989) [87] and Chester (1990) [88] consider the layers as **feature extractors**. They assign to the neurons of the first layer of a ffnn with two hidden ones the role of local feature extractors, with some of them being responsible for segmenting regions of the input space and others for identifying the characteristics of these regions. To the neurons of the second layer they assign the role of global feature extractors, believing that they use the local features identified by the neurons of the first layer for each region to create global features characterizing it. Like this, each region is represented by a combination of a few neurons and an update to the approximation concerning one region does not affect the neurons related to the rest of the input space. This is a theme that is central to this thesis.

The number of layers and the number of neurons at each one are thus important hyperparameters that significantly affect the ability of the net to generalize. They can be tuned before

the training process begins with a method that will be described in chapter 2.4.6. Optimizing the number of neurons of every layer is an indirect way of choosing the number of parameters of the NN. But, optimizing at the weight-level is also possible. One can initially choose a relatively large number of neurons and then directly eliminate the unecessary parameters during the training process to avoid overfitting.

Regularization in Neural Nets

Regularization is the most common way of handling the the bias-variance trade-off. This is usually done with the addition of an extra term to the cost function:

$$J(\mathbf{w}) = J_{av}(\mathbf{w}) + \lambda J_c(\mathbf{w}) \tag{2.57}$$

The first term is the familiar performance metric that measures how well the model fits the training data. The second term is the complexity penalty and is only dependant on the model's parameters. When:

- $\lambda = 0$: No regularization is applied and training proceeds as has been described
- $\lambda \to \infty$: The training samples are considered unreliable and the parameters are determined by the constraint alone

Usually an intermediate value is used that is responsible for controlling the bias-variance trade-off.

The most widely known regularizer is **weight decay**. In the case of weight decay equation 2.57 becomes:

$$J(\mathbf{w}) = J_{av}(\mathbf{w}) + \lambda \mathbf{w}\mathbf{w}^T = J_{av}(\mathbf{w}) + \lambda \sum_{i,j,n} (w_{ji}^{(n)})^2 \tag{2.58}$$

Complexity in that sense is measured by the weight values, i.e. the larger their magnitude the more complex the model. This type of penalty can be used in cases of models that are considered to be initialized with more weights than are necessary to approximate the underlying function. During training two types of weights are distinguished:

- weights that contribute to the modeling of the underlying function and thus take nonzero values
- weights that are unecessary and if left unattended would take arbitrary values leading to the problem of overfitting while only slightly reducing the training error. These weights are called **excess weights** and are eliminated by the regularizer

Weight decay can be interpreted as the negative logarithm of a zero-mean Gaussian prior distribution over the parameter vector. This can be viewed in the frame of MAP presented in chapter 2.3.4. Alternatively, weight decay can be viewed as a gradual weight decay of all weights during training according to:

$$\mathbf{w}^{new} = \mathbf{w}^{old}(1 - \epsilon),\ 0 < \epsilon < 1 \tag{2.59}$$

that permits only important weights to retain nonzero values due to their significant influence on the accuracy of the model. The parameter ϵ is connected to the error function in the following way:

$$J(\mathbf{w}) = J_{av}(\mathbf{w}) + \lambda \frac{2\epsilon}{a} \mathbf{w}\mathbf{w}^T \tag{2.60}$$

where a is the learning rate.

Weight decay is also named $L2$**-regularization**. Another closely related form of regularization is the $L1$**-regularization** defined as:

$$J_{L_1}(\mathbf{w}) = \sum_{i,j,n} |w_{ji}^{(n)})| \tag{2.61}$$

$L2$-regularization is known to shrink weights to low values but not exactly zero. $L1$-regularization on the other hand does exactly that, creating sparse parameter vectors on the process, i.e. parameter vectors that have a relatively small number of nonzero elements w.r.t. their dimension. This is desired when there are good reasons to believe that the underlying function can be modeled with only a few parameters.

Research has also been perform on exploiting second-order derivatives of the error surface to solve the bias-variance dilemma. For more information one can read the following on the method called **Optical Brain Surgeon (OBS)** introduced by Hassibi and Stork in 1993 [89].

Early Stopping

Setting the random noise apart, the training error generally decreases monotonically during the training of a neural model. The same is not true with the error on the validation set that has been seen to steadily decrease before beginning to increase again due to overfitting (figure 2.10). This happens because the net begins learning the noise contained in the training data instead of modeling the underlying distribution.

Early stopping is an alternative method to regularization exploits this fact by periodically measuring the error on the validation set and stopping the training process when this metric starts to increase, indicating that the model's generalizing abilities are at that point optimized.

Dropout

Weight initialization and the distribution of training samples in batches play an important role in the training of NNs but are probabilistic in nature. Therefore it is usual for engineers and researchers to employ ensembles of NNs that are trained independently to eliminate the uncertainty caused by the above factors. They then can use their mean output value for regression problems or the label chosen by the majority of NNs for classification problems. The disadvantages of this approach are the computational overhead and the increase of memory requirements.

Dropout was introduced by Srivastava et al. (2014) [90] as a cheap approximation to the ensemble approach and is essentially a regularization method. What it does is training the ensemble of all sub-networks that result from deactivating non-output neurons of a NN. It does that by randomly sampling binary masks for all these units of the network every time a new sample is used as input and information is forward propagated through the network. For each unit a new hyper-parameter, $p_{dropout}$, is defined that determines the probability of multiplying its output by zero during a random forward propagation procedure. If that happens then the respective neuron does not produces an output and is thus not trained for the corresponding sample. Since $p_{dropout}$ is chosen so that all neurons are left out at some point during training, the model is supposed to learn not to rely on a sub-network but is forced to have all of its possible sub-networks trained to cope with the potential shortage of trained paths. In contrast to standard ensemble methods, in dropout, all models share parameters with each other and most models are not trained at all since it is infeasible to use all possible sub-network combinations.

During inference, the ensemble is approximated by keeping all units active while multiplying each weight by the probability of not dropping its respective unit [91].

The method's overhead is negligible O(total number of neurons) and it is known to generally

perform well. Nonetheless it tends to increase the size of the model and the total training time like most regularization methods [7].

Batch Normalization

In section 2.4.4 the need for scaling the inputs of NNs was addressed. The same circumstances that lead to the need for input scaling may also appear between consecutive layers of a deep neural network (DNN). It is possible, for some random initialization of the weights of a layer, for the inputs of the activation functions of the layer to take large values causing the training process to become unstable. Moreover, the backpropagation algorithm, chapter 2.4.3, upgrades each layer's weights using the assumption that the statistics of the layer's inputs remain constant. Nonetheless, input statistics have been observed to vary between different batches of samples. Ioffe and Szegedy (2015) [92] named this phenomenon **internal covariance shift**. Internal covariance shift forces neural layers to continuously adapt to new input statistics and causes gradients to become unstable and training to slow down.

That is why they proposed **batch normalization**, a regularization method for solving this problem by normalizing the pre-activations of every neuron of a layer across the samples of a batch. The pre-activations of the neuron for all samples of a batch are placed in a set, have the mean value of set subtracted from them and results are divided by the standard deviation of the set. Yet, since information that may be useful is lost this way, two parameters, a scaling factor, gamma γ, and an offset factor, beta β, are introduced. The network learns optimal values for γ and β for each batch so that the normalization process becomes reversible. The steps for a neuron, with pre-activation $u^{(i)}$ for the i-th example of a batch of B examples, are given by the equations that follow:

$$\mu_b = \frac{1}{B}\sum_{i=1}^{B} u^{(i)} \tag{2.62a}$$

$$\sigma_b^2 = \frac{1}{B}\sum_{i=1}^{B}(u^{(i)} - \mu_b)^2 \tag{2.62b}$$

$$\hat{u}^{(i)} = \frac{u^{(i)} - \mu_b}{\sqrt{\sigma_b^2}} \text{ or } \hat{u}^{(i)} = \frac{u^{(i)} - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} \text{ for stability reasons when } \sigma_b^2 \text{ is very small} \tag{2.62c}$$

$$\hat{y}^{(i)} = BN(u^{(i)}) = \gamma u^{(i)} + \beta \tag{2.62d}$$

Batch normalization thus renders the model less sensitive to weight initialization and hyper-parameter tuning. Yet, if B is small the sampled mean and std are not representative of the actual distributions. For example, sequence models, like RNNs discussed in chapter 2.6, handle input sequences of variable lengths causing the batch size to decrease for long input sequences. Batch normalization might prove to be problematic for such cases, but it still remains a good solution for CNNs, chapter 2.5 [92, 93].

Layer Normalization

Ba et al. (2016) [49] proposed **layer normalization** as an alternative to batch normalization. Instead of normalizing the pre-activations of neurons across the examples of a batch, layer normalization normalizes the pre-activations for each sample across the neurons of a layer. If D is the

number of neurons of a hidden layer:

$$\mu_l = \frac{1}{D}\sum_{d=1}^{D} u_d \tag{2.63a}$$

$$\sigma_l^2 = \frac{1}{D}\sum_{d=1}^{D}(u_d - \mu_l)^2 \tag{2.63b}$$

$$\hat{u}_d = \frac{u_d - \mu_l}{\sqrt{\sigma_l^2}} \text{ or } \hat{u}_d = \frac{u_d - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}} \text{ for stability reasons when } \sigma_l^2 \text{ is very small} \tag{2.63c}$$

$$\hat{y}_d = BN(u_d) = \gamma u_d + \beta \tag{2.63d}$$

Note that γ and β parameters in the case of layer normalization are the same for all neurons of the same layer but different for summed inputs corresponding to different examples. The opoosite is true for batch normalization.

Batch normalization requires different processing during training and different during inference due to the change in batch sizes. The same is not true in the case of layer normalization as the model architecture never changes.

2.4.6 Model Selection

In chapter 2.4.3 a process of deciding the parameter values via a training algorithm was discussed. The success of a model is known to depend not only on the parameter values but also on the hyper-parameter ones. Such parameters are the tuning parameter λ of a regularizer, the number of layers of a NN, the initial value for the learning rate, the type of activation function used by the network's neurons, the type of regularization applied, etc. The values for these parameters must be decided before training initiates, but they must be chosen wisely because using a large learning rate or low-order polynomial w.r.t. the rest of the hyper-parameters' values and the problem at hand may render a model completely incapable of learning. The two main issues that must be dealt with is finding the way the search, or else **hyper-parameter exploration**, will be conducted and the data that it will use.

As far as hyper-parameter exploration is concerned, given that each hyper-parameter may take a set of values, a full search over the hyper-parameter search-space means considering every combination of the Cartesian product of these value sets. The computational cost of this method is usually massive due to the large number of hyper-parameters and of their respective possible values. Using prior information for the choice of the value sets and perform different searching rounds iteratively choosing every time the most promising value subsets based on the results of the previous round generally speed up the process but not drastically. On the other hand, pruning runs that don't seem promising and choosing the next combinations of values to be tested based on experience and prior knowledge are some tricks that are used by modern optimizers such as Ray [94] and Optuna [95] and usually improve the tuning time considerably.

The problem of choosing the dataset that will be used for hyper-parameter tuning is also tricky. Using the training set to derive value for the hyper-parameters is problematic, as these hyper-parameters may be optimum for it but this choice does not guarantee that the model will be good at generalizing to instances that have not been used in its training. In other words, this may cause the model to overfit the training data due to the design choices. A more reliable method would be to use different datasets for choosing the hyper-parameters and the parameters respectively. Therefore, using the validation instead of the training set to tune the hyper-parameters seems like a reasonable choice. Then the model is trained on the training set and, to test if the model has overfitted either of these too, a test set can be used to test if it can generalize. Usually, out of all

the samples that are not part of the test set, 20% is used as part of the validation set and the rest as part of the training set. This method is referred to as the **holdout method**, and is generally preferred in cases of big data sets and large models.

If training data is limited then one cannot afford using a significant part of it for tuning hyper-parameters. But, using a small validation set may result to inaccurate and noisy estimations. Instead, a method called **cross-validation** is used. The K-fold cross-validation method consists splitting the training set into K, equally sized, sample sets and performing K parameter exploration procedures, using every time $K-1$ sets for training and the held-out set for measuring the model's performance. This process is pictured in figure 2.16.

After all sets have used as held-outs for the test of a certain hyper-parameter value combination the average error over all K runs is calculated. A combination of hyper-parameter values is deemed than better than another if the corresponding average is error is lower. In the end, the best combination is used to initialize the model, hence the title **model selection**, and the training proceeds. Obviously the larger the value of K, the more precise the results and, in cases of data scarcity, the performance may be measured using only a single example every time. This variant is known as the **leave-one-out** method. Unfortunately cross-validation's complexity grows as K increases and applying it is usually feasible only for relatively small datasets and models.

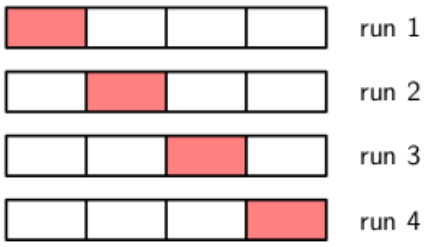


Figure 2.16. *4-fold cross-validation process [4]*

After decades of research there have been found combinations of values for sets of hyper-parameters that generally work for most of the well-known problems. Nonetheless, this does not mean that they are optimum in every case and that cross-validation is outdated. As research keeps focusing on new model types and unexplored problems keep on coming of cross-validation and its variants will always remain contemporary.

2.5 Convolutional Neural Networks

2.5.1 Invariances

In many applications it is required that inputs that differ with each other only due to the application of one or more transformations to one of them should yield the same output. For example this occurs in the field of image recognition, where an object in an image should be classified correctly irrespective of the angle, the distance and the lighting in which the picture was taken, like in figure 2.17. The same is true in the case of speech recognition where a certain word should be recognized as being the same no matter the person that spoke it or the background noise. Some applications thus require predictions to be **invariant** under one or more transformations of the input.

A way this difficulty might be handled is with the use of an abundance of data that contain examples of the results of the application of the various possible transformation. An example of this is a dataset that includes pictures of the same object taken at several different hours of a day. A model can use these examples to learn the invariance to natural lighting.

However obtaining such a number of examples is not always possible and alternatives are thus needed. These alternatives are classified as follows [4]:

- Augmenting the dataset with manually transformed samples in order to create the conditions

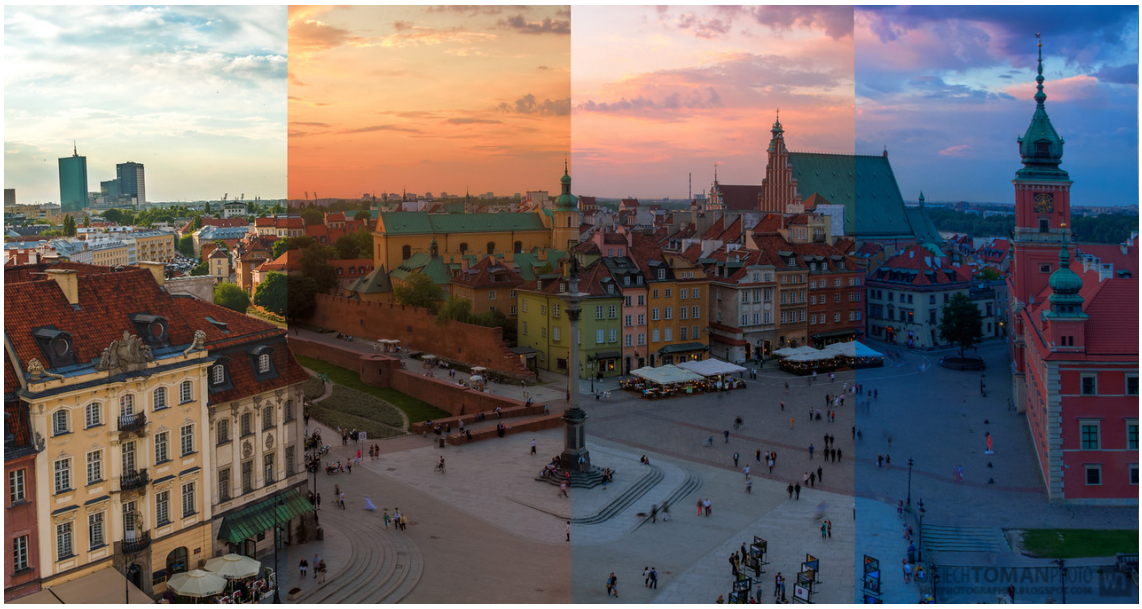


Figure 2.17. *Images of the same place taken at different hours of the day. Figure from* `http://hdr-photographer.com/2014/10/different-times-of-day/`

described above. This method is known to effectively improve generalization but can lead to considerable computational burden

- Using a regularizer that penalizes changes to the output when predicting the label of an input that has only been transformed. This approach and the ways of applying it are closely connected to the first alternative

- Extracting the invariant features as part of the preprocessing process and using them as input during training. The model may then be able to generalize even to cases not seen during training. Yet, some useful features might be left out during the extraction process

- Incorporating these invariances into the model's structure. This alternative will be the main focus of this chapter as it is closely connected to the central theme of utilizing prior knowledge to adapt the model's architecture

2.5.2 Priors in Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [96, 97] are one of the most successful cases of incorporation of prior knowledge into a model's structure. Even though CNNs have been used in a variety of fields it is instructive to first consider them under the umbrella of computer vision as they were initially conceived as image processing models. An image is usually represented as a grid of numbers of dimension $D \times D$, which record pixel intensities. In the case of coloured images the grid of numbers becomes a grid of vectors containing values for the three colour channels, red, green and blue corresponding to each pixel.

The matter of invariances in image datasets was addressed previously. When such constraints are known beforehand it is usually beneficial to incorporate these prior beliefs into the model's structure. This takes the form of a probability distribution over the parameters of the model encoding the beliefs about about what models would be reasonable before looking at the dataset. In the case of CNNs, these priors result from the properties of visual data and from invariances that are known to be found in them. Moreover, priors may be considered as being weak or strong depending on the degree of concentration of the probability density function in the prior.

Weak priors are characterized by low levels of concentration in their density functions and allow parameters to relatively unconstrained. Strong priors, on the other hand, play a bigger role to parameter value formation.

Sparse Connectivity

A well known property of images is the existence of a strong correlation between nearby pixels, since they usually belong to the same object. Yet, each neuron of a FFNN would unnecessarily receive the entire image as input and would then have to learn to focus on individual input subspaces. This prior can be incorporated by forcing neurons to focus on areas much smaller than the entire input and thus naturally extract local features. This property is referred to as **sparse connectivity**.

In multi-layer architectures these features can be combined in later layers to create higher-order features and result in yielding global information about the picture as a whole. For example, when processing an image with thousands of pixels, one can use a square-shaped receptive fields containing only tens or hundreds of parameters. These are called **kernels**. The use of kernels reduces the memory requirements of the model and the computational burden, i.e. in the case of D_{in} inputs and D_{out} outputs and a kernel of k parameters, runtime is reduced from $O(D_{in} \times D_{out})$ to $O(k \times D_{out})$. This type of prior is considered to be an infinitely strong prior since the rest of the parameters are essentially forced become zero.

Weight Sharing

An invariance that must be addressed is translation invariance. A feature may be found in any part of the image and, consequently, the kernel responsible for extracting it should scan the entire image searching for this feature. The implementation of this prior is done by imposing weight sharing among kernels whose combine receptive fields covers the entire image. Weight sharing forces them to detect the same feature across the picture.

Because of the weight sharing the combined computations performed by these kernels are equivalent to a convolution operation applied to the image by the shared kernel. Therefore, instead of using multiple identical kernels, one can use a single one that is shifted as is ordered by the convolution. A toy example is shown in figure 2.18. Note that this is the result of an input-kernel convolution, before an activation function is applied to it. Weight sharing does not change the runtime since the computations are either way performed in parallel. Yet, it further reduces the storage requirements to k parameters with k usually being much smaller than D_{in}. Because of the constraint applied on the parameters, weight sharing is considered another infinitely strong prior.

Sub-sampling

The operation that usually follows feature extraction is sub-sampling performed in the form of pooling. During pooling, a group of outputs of the feature extraction process that have resulted from applying the kernel to adjacent image areas is sub-sampled with the application of a pooling function. The most well-known pooling function is the max function that chooses the input with the largest value. Another one is the averaging function that outputs the average of its inputs. Pooling makes the representation approximately invariant to small translations of the input, applying the prior belief that one is not interested at the exact spot where a feature is found but is more concerned about whether it exists or not. This is also an infinitely strong prior.

The realization of these priors thus can substantially reduce computational and storage requirements. But it is also possible that the assumptions made might be inaccurate, like in the case

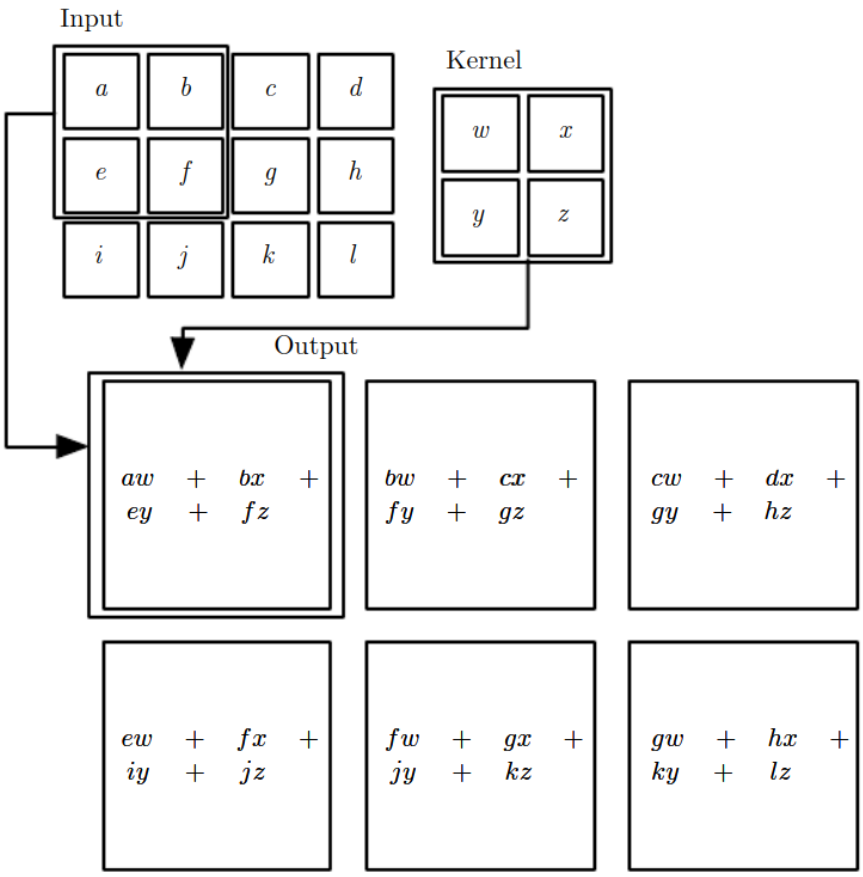


Figure 2.18. *Example of a convolution in two dimensions without kernel flipping. Outputs for the calculations of which only a part of the kernel would be used are not shown [7]*

where information about the specific locations of objects is needed. In that case, using pooling would increase the training error. In other words, these priors, like many others, are known to cause **underfitting** [4, 7].

2.5.3 The Model

As noted by Goodfellow et al. (2015) [7], convolutional neural networks are neural networks that use convolution instead of general matrix multiplication in at least one of their layers. To better understand how the modules of a CNN are combined the example presented in figure 2.19 will be followed. A CNN module is a convolutional layer followed by a pooling one. The convolutional layer is composed of a set of kernels each of which is responsible for detecting a feature. Each kernel is comprised of a set of weights, the number of which equals the size of the kernel's receptive field plus one added to account for the bias parameter, followed by an activation function. In the example shown in the figure, the first convolutional layer consists of $k = 4$ kernels with a receptive field of 5×5 each. Notice that the dimension of each feature map smaller than the input's the same way it is in 2.18. After the activation function is applied a sub-sampling layer follows. The neurons of the first sub-sampling layer of the example has receptive fields of 2×2, hence the dimension of the corresponding feature map. The outputs are multiplied by trainable coefficients, increased or decreased by the addition of a trainable bias and transformed by an activation function.

The second convolutional layer uses 12 kernels with 3-dimensional receptive fields of $4 \times 5 \times 5$ that span across the 4 feature maps of its input. The second sub-sampling layer is similar to the

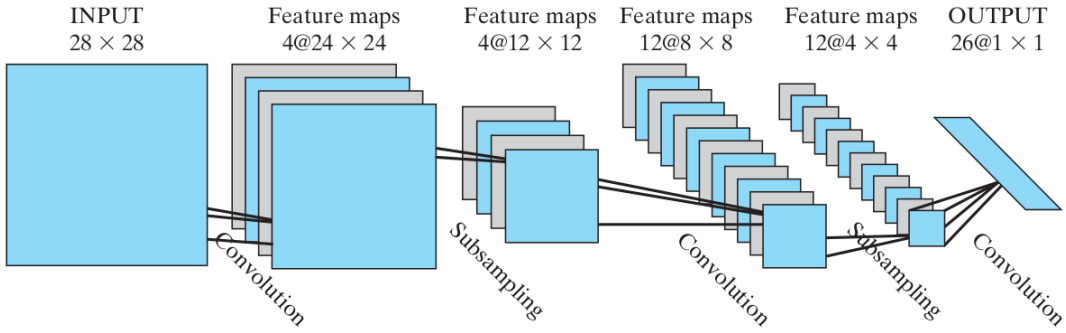


Figure 2.19. *Example of a convolutional neural network [8]*

first but with 12 feature maps instead of 4. The output layer is a convolutional one with 26 kernels receptive fields of $26 \times 4 \times 4$. This essentially is a feed-forward layer.

The model has approximately 100000 connections but only 2600 trainable parameters. This shines light on the importance of utilizing priors when designing a neural architecture. The model is trained with backpropagation that deals with weight sharing by using, for the update of a certain variable, data from all computations that were performed with its participation.

2.5.4 Inspiration From Biology

CNNs were one of the first deep learning models that were trained successfully with backpropagation. It is thus a great surprise that the creation of this model was inspired by scientific research about the biology of the human visual system. $\mathbf{V}_1$, an area of the brain known to be part of this system has be found to be arranged in a 2-dimensional grid of neurons that process regions of the visual input which spatially correspond to their position in the grid. Some of them are called **simple cells** and perform feature detection with the use of linear functions. Other are called **complex cells** since, apart from that, are invariant to small shifts in features' positions and to changes of the lighting.

Using the way human brain works to discover biases and methods of implementing them is a central point of this these.

2.6 Recurrent Neural Networks

Data in some machine learning problems comes in the form of a sequence of vectors $\{\mathbf{x}_i\}_{i=1}^{\tau}$. For example this is the way the words of a text or the prices of a certain stock during a specific time interval are represented. The length of these sequences may vary from sample to sample. Feed-forward and convolutional neural networks require the maximum sequence length to be predetermined and cannot extrapolate to bigger sequences. For this reason **recurrent neural networks (RNNs)** [98] were introduced. RNNs were made possible because of exploiting the weight sharing prior that enables them to generalize to sequence lengths that were not observed during training by using the same set of weights for every vector of the sequence.

2.6.1 The Model

In order to process an vector of the input sequence, information from the vectors that precede it is usually necessary. RNNs ensure the existence of such information by maintaining a state vector that is updated regularly. Specifically, at a random step t, an RNN takes two inputs, the next vector in the sequence $\mathbf{x}_t$ and the state vector computed in the previous step, $\mathbf{h}_{t-1}$. The letter

h is used to indicative of the fact that this vector results from hidden units of the network. The equation for the state vector is the following:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\vartheta}) \tag{2.64}$$

where $\boldsymbol{\vartheta}$ is the parameter vector and f is a nonlinear function called a **transition function**. Notice that the parameter vector does not depend on the step parameter due to the weight sharing technique applied. Therefore, both $\mathbf{x}_t$ and $\mathbf{h}_t$ dimensions' are constant and independent of t.

The most famous RNN version is the one presented in figure 2.20 with the method of graph unfolding. It is described by the following equations:

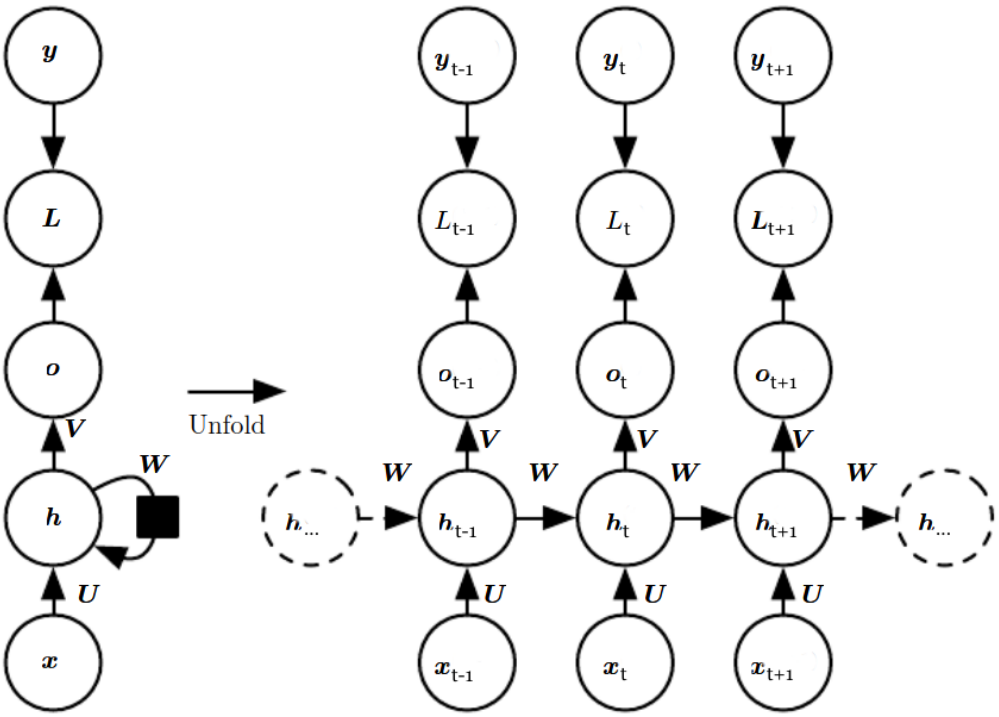


Figure 2.20. *Example of a recurrent neural network presented with the method of graph unfolding. The arrows represent the flow of information at a random step t [7]*

$$\mathbf{u}_t = \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b} \tag{2.65a}$$

$$\mathbf{h}_t = g(\mathbf{u}_t), \text{ where } g \text{ is a nonlinear function} \tag{2.65b}$$

$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{c} \tag{2.65c}$$

$$\hat{y}_t = s(\mathbf{o}_t) \tag{2.65d}$$

Equations 2.65a and 2.65b together make equation 2.64. This RNN computes an output at every time step with the use of 2.65c and 2.65d. A usual choice for an output activation function is the softmax (equation 2.29) since this RNN type is usually used to pick an element from a predetermined set at each step. In language modeling the set is the vocabulary and the elements are words. Notice again that the weights do not depend on the step parameter and that, before forward propagation initiates, the starting state $\mathbf{h}_0$ first has to be specified.

The RNN presented in figure 2.20 maps an input sequence to an output sequence of the same length. Each input sequence $\{\mathbf{x}_t\}_{t=1}^{\tau}$ comes paired with a set of correct predictions $\{y_t\}_{t=1}^{\tau}$, while the model's erroneous predictions are denoted by $\{\hat{y}_t\}_{t=1}^{\tau}$. At each step the negative log-likelihood

is computed and the total loss equals the sum of these individual losses:

$$L = -\sum_{t=1}^{\tau} logP_{model}(y_t|\mathbf{x}_1, \cdots, \mathbf{x}_t) \tag{2.66}$$

In order to compute the gradient w.r.t. a parameter of a RNN one, like in the case of CNNs, has to account for all computations in which it took part. Since each parameter participated in one computation at each step, τ computations have to be considered and the contribution of each one to the gradient is computed following a direction opposite to the one indicated by the horizontal arrows in figure 2.20, i.e. from right to left. Due to the sequential nature of the interaction between the RNN and the input nor forward neither backward propagation can be parallelized.

2.6.2 Teacher Forcing

Another well-known RNN type uses as input the output that it generated in the previous step. This is for example the case with language generation models that feed the previously generated word as input for the next step so that the model can use it to choose the next word.

Such models can be trained in two ways. First, one can feed the model with the actual outputs of the previous step to approximate the inference process as closely as possible. Another way is to provide the model with the ground-truth inputs irrespective of the model's generated outputs. One benefit of this method is its mathematical soundness, since it implements the following maximum likelihood equation:

$$\log p(\mathbf{y}_1, \mathbf{y}_2|\mathbf{x}_1, \mathbf{x}_2) = \log p(\mathbf{y}_1|\mathbf{x}_1, \mathbf{x}_2) + \log p(\mathbf{y}_2|\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}_1) \tag{2.67}$$

In addition to that it allows the parallelization of the training of architectures that don't use the previous hidden states, h's, as inputs, i.e. models for which the matrix $\mathbf{W}$ equation in 2.65a is a null matrix.

In order for the model to get used to being used in a closed-loop mode, Bengio et al. (2015) [99] proposed mixing the two training methods using each one for a part of the total updates [7].

2.6.3 Deep RNNs

Thanks to the parameter sharing technique, the total number of trainable parameters of a RNN is $O(1)$ w.r.t. the input length. This allows the use of deeper RNN architectures than the one presented previously. Essentially, an RNN implements two functions:

- the one that uses the current input and the previous hidden state to compute the new one, equations 2.65a and 2.65b
- the one that uses the current hidden state, and in some case the current input, to compute an output, equations 2.65c and 2.65d

These were previously implemented with a single neural layer each. Yet, like in the case of ffnns and CNNs, it is also possible to use more than one layers in order to benefit from the ability to create and use features of multiple orders.

Of course deep architectures tend to make training harder due to the vanishing gradient problem, chapter 2.4.4. Nonetheless one can use methods like ReLU and skip connections to improve training speed and quality.

2.6.4 Bidirectional RNNs

In figure 2.20 all horizontal arrows point from left to right, indicating that, for the computation of the output at a random step, information only from previous inputs and the current one might be used. But, in some problems information about the entire input sequence is needed. For example, in part-of-speech tagging a word may function as an adjective or a noun depending on whether the following word is a noun or not. The word "homeless" is considered an adjective in the phrase "providing houses for the homeless" and a noun in the phrase "providing houses for the homeless people".

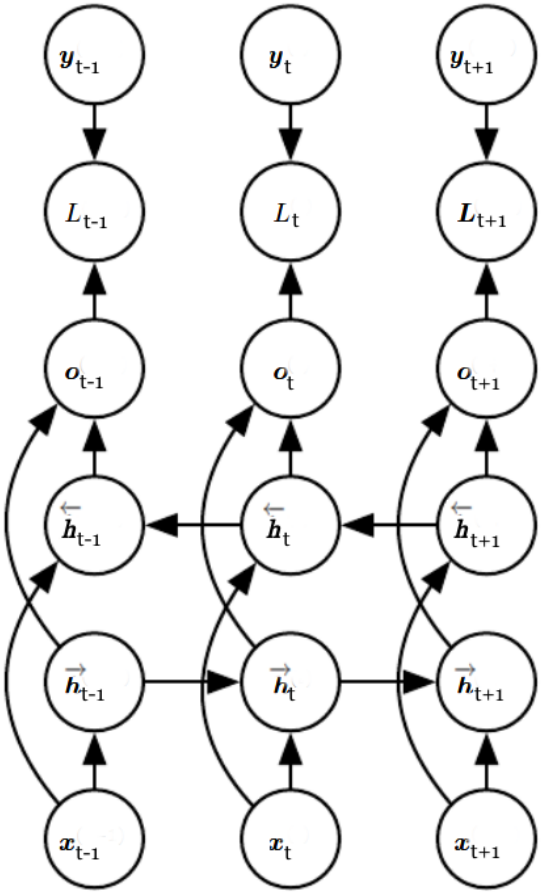


Figure 2.21. *Example of a bidirectional RNN presented with the method of graph unfolding [7]*

Bidirectional RNNs (BiRNNs) [100] were created to provide a solution to this problem. In essence, they are made of two RNNs that process the input sequence following opposite directions, one from left to right and the other from right to left. The respective state vectors $\overrightarrow{h}_t$ and $\overleftarrow{h}_t$ provide the necessary information for the computation of the output.

2.6.5 Encoder-Decoder Architectures

The RNN type presented thus far maps input sequences to output sequence of the same length. It is also possible to omit the intermediate outputs and map an input sequence to a single output vector that can be created after processing the last element of the input. Yet it is impossible for it to map an input sequence to an output sequence of variable length, which is is required in fields like machine translation, speech recognition and question-answering. In order to satisfy this need, the **encoder-decoder** or **sequence-to-sequence model** was proposed by Cho et al. (2014) [101]

and Sutskever et al. (2014) [102]. The model employs two RNNs, one is called an encoder and the other a decoder.

The encoder is responsible for transforming the input sequence into a context vector c that properly summarizes the first. The dimension of c is constant and independent of the length of the input sequence. The elements of the input sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_{\tau_{in}})$ are usually represented by vectors of numbers that belong to dictionaries that may contain vectors for hundreds of thousands of elements. Even though these vectors are made to characterize their respective elements, they are only fully defined by taking into consideration their use in the input sequence. For example, the meaning of some words depends on the context in which they are found. The word "cold" has two different meaning in the sentences "He went out without a jacket, so he caught a cold" and "Its cold in here, turn on the heat". Consequently,, when performing tasks such as neural machine translation (NMT), it is necessary to create representations of the input sequence that take the context into consideration. The role of the encoder is thus to produce **contextual representations** of the input sentence [9].

The decoder is in charge of generating the output sequence $y = (y_1, y_2, \cdots, y_{\tau_{out}})$, while being conditioned on c. Notice that it τ_{out} is not necessarily equal to τ_{in}. Conditioning can be performed by simply adding another to term in equation 2.65a. At each step the output of the previous step is fed to the decoder as input. This property renders the model **auto-regressive**. Other than c there is no other constraint that the vector dimensions used by one RNN impose to the vector dimensions used by the other.

The system is trained to minimize the negative log likelihood:

$$L(\boldsymbol{\vartheta}) = -\sum_{i=1}^{N} \log P_{\boldsymbol{\vartheta}}(y_1^{(i)}, y_2^{(i)}, \cdots, y_{\tau_{out}}^{(i)} | \mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \cdots, \mathbf{x}_{\tau_{in}}^{(i)}) \tag{2.68}$$

where N is the total number of training samples.

A serious disadvantage of this architecture is that c, which is of constant dimension, must summarize a sequence of variable length. As input length increases this model's performance is known to drop. This problem will be addressed with the help of the attention mechanism discussed in chapter 3.7.

2.6.6 The Problem of Long-Term Dependencies

The problem of vanishing gradients is known to arise when training deep neural network architectures. A similar effect is caused in RNNs due to the big length of some input sequences. This causes gradients to vanish slowing training down or increase uncontrollably rendering training impossible. To see why this is true, consider the case of an RNN without an activation function defined in equation 2.65b. The input is also omitted for simplicity. Then it is true that $\mathbf{h}_t = \mathbf{W}\mathbf{h}_{t-1}$. Unrolling this inductive relation to the first step results to $\mathbf{h}_t = \mathbf{W}^t\mathbf{h}_0$. If $\mathbf{W}$ can be decomposed to $\mathbf{W} = \mathbf{Q}\Lambda\mathbf{Q}^T$, where $\mathbf{Q}$ is orthogonal, then $\mathbf{h}_t = \mathbf{Q}\Lambda\mathbf{Q}^T\mathbf{h}_0$. Raising Λ to the power of t causes eigenvalues smaller than one to decay to zero and eigenvalues bigger than one to increase exponentially.

Unfortunately staying in the region of parameter values that guarantees that gradients will neither vanish nor explode is impossible. Bengio et al. (1993) [103] and Bengio et al. (1994) [104] proved that gradients corresponding to long-term relationships will inevitably by exponentially smaller than the gradients of short-term relationships.

However various techniques for assuaging this problem have been proposed. Some of them are:

- Using skip-connections through time that allow signals to bypass d time steps in order to capture longer dependencies. This method does not solve the vanishing or exploding gradients

problems; it simply delays them.

- Use leaky units; units with linear self-connections and weights near one that allow safe passage of gradients.
- Long short-term memory (LSTM) model, Hochreiter and Schmidhuber (1997) [105]: using the idea of linear connectivity to allow gradients to pass and of **gating** implemented by neurons with sigmoidal activation functions to allow the model to decide automatically which pieces of information will be granted passage to the next computational stage. Some pieces of information are useful for some units and in some steps while other pieces might be irrelevant. Gating implements the prior that it is beneficial for the model to be able to focus only on what is known to be useful and not on information that might be unnecessary or simple noise. Input gating improves the generalizing abilities of the model as it is able to ignore signals it has never seen before which are likely to be irrelevant. These signals would introduce noise interfering with the computations performed by the model. Moreover, the forget gate erases a part of the state vector increasing the available space for storing useful data for a potentially large number of steps.

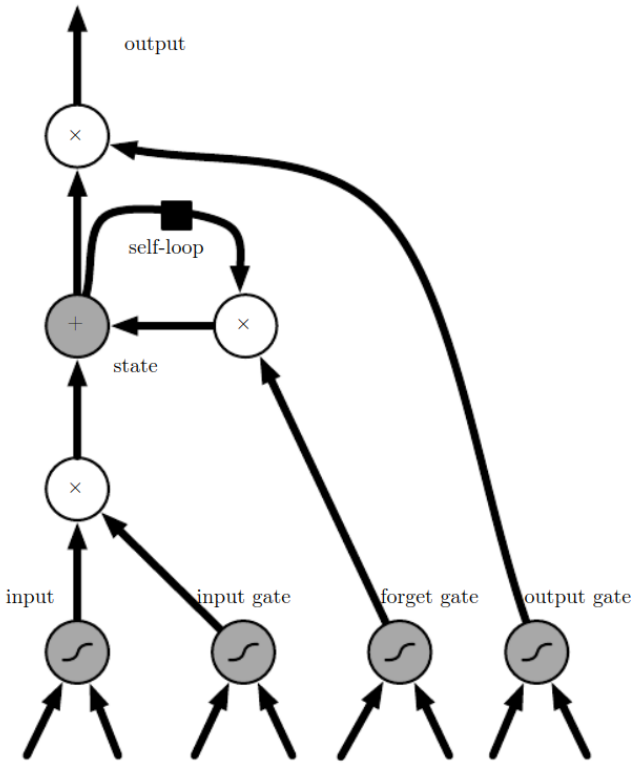


Figure 2.22. *Architecture of the LSTM model [7]*

- Gradient Clipping [76]: a common form of gradient clipping consists of simply clipping $\nabla_{\mathbf{w}} L$ before the update:

$$\text{if } \nabla_{\mathbf{w}} L > u \Rightarrow \nabla_{\mathbf{w}} L \leftarrow \frac{u \nabla_{\mathbf{w}} L}{||\nabla_{\mathbf{w}} L||}$$

This leaves the gradients' direction unchanged and sets its magnitude to a user-defined value. Gradient clipping mitigates the exploding gradients problem and helps in cases of error surfaces with irregular and steep slopes.

Chapter 3

Natural Language Processing

3.1 Introduction

Much of the information that is used by humans is stored in the form of text. People continuously use written language in their daily lives, whether that is for communicating via emails, deciding which movie to watch with the help of relevant critics or for ordering products online using chat-bots. The research community has been studying for decades methods that allow computers to facilitate these interactions between humans and natural language. Like this, the research field of natural language processing **(NLP)**, the understanding, analysis and use of human language by computers, was born. In contrast to programming languages that have their symbols and words precisely defined, natural language tends to be vague and ambiguous and as such it demands a different approach, that will be the main focus of this chapter [7, 106].

Generally, a NLP system is a pipeline of modules that transform, provide structure to, extract information from and may even generate text. Methods that dominated the NLP pipelines during the 20-th century were rule-based, i.e. text was processed with the use of predetermined rules that were implemented by efficient algorithms like finite state automata [107]. Modern NLP has mostly replaced them with statistical and machine learning methods that require minimum human labor and benefit from the massive, in comparison to the past, available computational power and storage spaces. The use of such methods has been possible only thanks to the size of textual **corpora**, vast amounts of textual data that are available online, like the Wikipedia corpus [106].

3.2 Text Normalization

Early on in the pipeline, text is formatted in a way that allows the following subsystems to concentrate only on extracting meaning from it. **Text normalization** is a collection of transformations applied on textual data to convert it into such a standard form. There are three main sub-tasks that make up text normalization:

- word tokenization: the task of separating words from running text
- normalizing word formats: the task of placing words in a predetermined format
- sentence segmentation: the task of splitting up sentences

A NLP system maintains a list of words for which it stores, and sometimes updates, relevant information. This list is called the **vocabulary** of the system. Training sets are usually made of running text comprised of words, which are named **tokens**. A vocabulary can be predetermined and set before the parsing of the training set is performed. Alternatively, one can parse the training set and, after a preprocessing step, use the set of words that occur more frequently than τ times, where τ is a user-defined threshold, as a vocabulary. Once a vocabulary is set, there is a possibility

of encountering words that do not belong in it. Such words are known as **out-of-vocabulary (OOV) words** and are usually represented by the model by a special token, <unk> [9].

3.2.1 Word Tokenization

A white space-based separation is usually problematic. Phrases like "United Arab Emirates" should count as single entries and not be divided into separate words. Another problem is the handling of punctuation that may occur inside words, like in the cases of "w.r.t." and "cap'n", in number expressions, e.g. 5.000,62. These issues highlight the need for deploying trustworthy **Named Entity Recognition (NER)** systems, that dig out words or phrases that correspond to real-world entities, dates, names etc. These are very useful, for example, in biomedical applications, where entities such as chemical names, genes and cells are usually described. There are also character-level tokenizers that separate words into their constituting letters.

Choosing a word-based optimizer can lead to big vocabulary sizes, while increasing the threshold τ in order to decrease its size leads to large numbers of OOV words. On the other hand, character-based tokenizers, even though they reduce the number of OOV words, are known to create long sequences and less meaningful individual tokens. A third category of tokenizers that seeks to balance the first two approaches are sub-word-based tokenizers. Sub-word-based tokenizers split words into their constituting parts. The word "playing", for example, might be formed by two vocabulary tokens, "play" and "-ing". The suffix token has a meaning of its own that slightly alters the meaning of the root word. This effectively decreases the number of OOV words, since a rare word will be split into frequently occurring morphemes or other sub-words, like "-er". Usually, the resulting sub-words are the biggest ones possible, which means that a word will undergo the minimum number of splits possible contributing to the decrease of the sequence length. The use of sub-word tokenizers also reduces the vocabulary size since storing every word of the text is avoided [9].

3.2.2 Byte-Pair Encoding

A famous greedy sub-word-based tokenization algorithm is **Byte-Pair Encoding (BPE)**. BPE creates the vocabulary using the training set. It first adds a special symbol </w> at the end of each word. The token learner starts by splitting up words into their letters and considers each letter to be a byte. It then counts the frequency of adjacent byte pairs and merges the most frequent one into a new byte. This process is repeated until an iteration limit or a token limit is achieved. Take for example a dictionary of a toy text that stores the number of appearances of each word: d={"old</w>": 7, "older</w>": 3, "finest</w>": 9, "lowest</w>":4}. The algorithm splits the words into letters and counts the number of each pair's occurrences:

- It. 1: "e" + "s" → ("es": 13)
- It. 2: ("es": 13) + "t" → ("est": 13), ("es": 0)

  Notice that the count corresponding to the byte "es" is reduced to 0, since all of its appearances are now included in the bigger byte "est".
- It. 3: ("est": 13) + "</w>" → ("est</w>": 13), ("est": 0)
- It. 4: "o" + "l" → ("ol": 10)
- It. 5: ("ol": 10) + "d" → ("old": 10), ("ol": 0)
- It. 6: "fin" is found 9 times in the text but only as part of the same word; so the merging process stop here.

At the end of the iterations the word "older</w>" for example is represented by less tokens (4) than at the beginning (6).

Note that with the addition of the symbol "</w>" the byte "est</w>" in "lowest</w>" is separated from "est" in "estimate", which is the desired behaviour since their meanings are distinct. But most importantly, "</w>" enables the decoding of the token sequence as it indicates the ending of each word.

When a tokenizer is trained on a training set it can be used to tokenize texts it has never seen before. To do that, a token parser first searches for each word in the vocabulary. If it is in it then no further action is needed. If it is not, it segments it in a way that uses the minimum number of vocabulary tokens ensuring that all sub-words will be meaningful [108].

3.2.3 Word Normalization

Word Normalization is the task of placing tokens in a standard form and of choosing a single form for words with the same meaning, like "gettin'" and "getting". Word normalization includes methods such as lemmatization and stemming [9].

Lemmatization is the process of locating words that are in different forms but have a common root and of removing sub-word parts that differentiate them, e.g. "playing" and "plays". This process is closely related to the part-of-speech (pos) tagging task, i.e. the task of determining the part of speech on which each word belongs. As a task, it is part of the field of **Syntactic and Morphological Analysis**.

Lemmatization may require the transformation of a word. **Stemming** is a much simpler and more naive approach that usually entails cutting off word suffixes. A famous algorithm that implements stemming is the Porter Stemmer [109].

3.2.4 Sentence Segmentation

Sentence are relatively small independent text fragments and thus many models process text sentence by sentence. This renders the task of **sentence segmentation** necessary in most NLP systems. Sentence segmentation is based on the following punctuation: periods, question marks and exclamation points. A problem with periods is that they are ambiguous, e.g. the period in the word "Mr." does not indicate the ending of a sentence. This is why sentence segmentation and word tokenization are sometimes performed jointly. A rule-based or a neural model determines whether a period is used to end a sentence or as a part of a word and then the two tasks proceed unhindered [9].

3.3 Language Models

A variety of NLP applications as well as the training of most neural models for NLP are based on the task of **language modelling (LM)**, i.e. the task of defining probability distributions over sequences of words, characters or bytes in natural language. These probabilities can be used for example to decipher the most probable next word in a noisy speech signal, to determine the way a wrongly spelled word will be replaced, to choose the most probable translation of a sentence or even to generate new text.

3.3.1 n-grams

Language models (LMs) are in charge of assigning these probabilities based on prior experience, that is, based on their training on a corpus consisting of a collection of documents. A language

modelling task is the task of estimating the probability of a sequence of words $w_1 \cdots w_n = w_{1:n}$. Suppose that the probability of a word following exactly after a phrase, symbolized as h and denoted as $p(w|h)$, e.g. $p(\text{cars} \mid \text{I prefer bicycles over})$. The computation of this quantity comes down to computing the following conditionals

$$p(w_{1:n}) = \prod_{k=1}^{n} p(w_k|w_{1:k-1}) \tag{3.1}$$

A language model could estimate the conditionals by counting the occurrences of $w_{1:k}$ in the corpus, $C(w_{1:k})$, and divide them by the occurrences of the context, $w_{1:k-1}$, $C(w_{1:k-1})$, $k \in \{1, \cdots, n\}$:

$$p(w_k|w_{1:k-1}) = \frac{C(w_{1:k})}{C(w_{1:k-1})} \tag{3.2}$$

The problem is that there are so many possible word combinations that even the largest corpora main not contain occurrences of certain long word sequences. The simplest possible LM, the n-**gram**, solves this issue by using the **Markov Assumption**, i.e. that the outcome of a random experiment depends only on the outcomes of a constant number, N, of immediate past experiments:

$$p(w_n|w_{1:n-1}) \approx p(w_n|w_{n-N+1:n-1}), \forall n \in \mathbf{N} \tag{3.3}$$

Equation 3.3 is the equation of the N-gram language model. For example, for a 2-gram (bigram) LM equation 3.1 becomes:

$$p(w_{1:n}) = \prod_{k=1}^{n} p(w_k|w_{k-1}) \tag{3.4}$$

Given that data is abundant, the bigger the value of n the more accurate the approximation in equation 3.3 at the cost of increased computational complexity.

A few notes on language modelling: In LM, sentences are usually augmented with special beginning, "<s>", and ending, "</s>", tokens, e.g. "<s>Hello there! How are you?</s>". Moreover, the probabilities are computed in log format as to avoid possible numerical overflow due to the extremely small probability values involved in the computations. Finally LMs handle OOV words by replacing them with "<unk>" tokens and estimating probability distributions for them [9].

3.3.2 Evaluating Language Models

Language model evaluation methods are categorized into extrinsic evaluation methods and intrinsic evaluation methods.

In **extrinsic evaluation** the LM is incorporated in an application and the performance of the overall system is measured. The best LM is considered to be the one that improves the performance of the application the most.

Using the LM in an application is usually a computationally inefficient way of evaluating it. That is why many times **intrinsic evaluation** methods are used, that measure the model's performance based on its own outputs. The respective metrics can be easily obtained but a model that does well by some metric does not necessarily perform well when incorporated in an application. An easy way to evaluate a LM is by computing the probability it assigns to a test set that it has never seen before. A LM that assigns a high probability to a test set is consider to have better knowledge of the language than one that assigns a lower probability to it.

Perplexity

A common practice is to avoid using probability values, but instead use a metric called **perplexity (PPL)**. Perplexity is the normalized inverse probability of a test set:

$$ppl(W) = p(w_1 w_2 \cdots w_L)^{-1/L} = \sqrt[L]{\frac{1}{p(w_1 w_2 \cdots w_L)}} \overset{chain}{\underset{rule}{=}} \sqrt[L]{\frac{1}{\prod_{i=1}^{L} p(w_i | w_1 \cdots w_{i-1})}} \tag{3.5}$$

For a bigram model then it is true that:

$$ppl(W) = \sqrt[L]{\frac{1}{\prod_{i=1}^{L} p(w_i | w_{i-1})}} \tag{3.6}$$

A model that assigns a text a high probability will lead to a low perplexity value on the same text. Perplexity can also be viewed as the weighted average branching factor of a language and is directly linked to the notion of entropy. For more information on the matter see [9].

3.4 Representing Words in NLP

A problem that has not been addressed yet is defining a way in which words will be represented so that they can be processed by NLP models. One could possibly use **one-hot vectors** to represent words, i.e. vectors of dimension equal to the size of the vocabulary, $|V|$, that represent a word by containing an ace in the corresponding position while setting the rest of the $|V| - 1$ positions to zero. A disadvantage of this approach is that these vectors fail to provide information about the meaning of the word, but simply inform the model of its existence. Yet, due to the ambiguity of natural language, it is not clear what constitutes a word's meaning, let alone how that could be distilled into a numerical vector.

Two words are considered to by similar of they are usually found in similar contexts and have similar meanings. But that is not the only way in which words might be connected to each other. For example, the words "road" and "car" are clearly related but not deemed similar. It was Joos (1950) [110], Harris (1954) [111] and Firth (1957) [112] who first used the contexts in which words are found to describe their meaning. The intuition was that words that tend to occur in the similar environments have similar meaning, also known as the **Distributional Hypothesis**. Thus, the idea of representing a word using vectors that are produced by taking advantage of the distribution of its neighbouring words was created. These vectors are called **embeddings**, because the meaning of the word is embedded into a vector space.

3.4.1 Sparse Embeddings

Long, sparse embeddings are created mainly by counting the number of times words appear in every possible context. They are usually stored in **co-occurrence matrices**. The simplest approach is to consider words that are found in the same documents to be semantically related. One can create a **term-document matrix TD** [113] with each position TD_{ij} storing the number of times the corresponding word w_i is found in the respective document d_j. These vectors $\{TD_{ij}\}_{j=1}^{|D|}$, $i \in \{1, \cdots, |V|\}$ can then be used to represent words.

Alternatively, one can use a **term-term matrix TT** of dimensions $|V| \times |V|$. A window of c words is determined and the number of times each word w_j is found in the context $\{-c, \cdots, c\}$ of the target word w_i is measured and stored in TT_{ij}. Usually, context values range from 2 to 5, with smaller values known to attribute more importance to the syntactic role of the words and larger values to their semantic roles.

A problem with using simply using counts to represent words is that the importance attributed to words like "the" or "and" that are found in all contexts and documents is disproportionate to their actual semantic gravity. Therefore the **term frequency–inverse document frequency (tf-idf)** measure is employed. The tf-idf measure is the product of two terms:

- term-frequency (Luhn, 1957) [114]:

$$tf_{t,d} = \log_{10}(count(t, d) + 1) \tag{3.7}$$

  where the logarithm is additionally employed to avoid a situation where a word that is found a 100 times more frequently than another one is deemed a 100 times more relevant

- inverse document frequency (Sparck Jones, 1972) [115]:

$$idf_t = \log_{10} \frac{|D|}{df_t} \tag{3.8}$$

  where df_t is the number of documents that contain the term t

To fill in each position of the respective matrix the two terms are multiplied together:

$$tf - idf_{t,d} = tf_{t,d} \times idf_t \tag{3.9}$$

3.4.2 Dense Embeddings

Short, dense embeddings, usually of dimension in the range between 50 and 1000, have been found to outperform long, sparse embeddings in nearly every NLP task. Possibly this is because the smaller number of parameters reduces the model's variance, thus improving its ability to generalize. This section will focus on **skip-gram with negative sampling (SGNS)**, a simple, yet effective method, of creating dense word embeddings proposed by Mikolov et al. (2013) [116]. Their intuition was to train a neural classifier on the binary classification task of predicting whether a word is found in the context of the current word or not and then use the trained weights as embeddings for the two words. A technique that they employed and played a crucial role in increasing the efficiency of their model was the use of running text to create the supervised learning task. The classifier parses running text, selects a window of words, uses the word in its center as input and its context words as golden answers to the binary classification problem. This variant of supervised learning is called **self-supervision** and it essentially saves the researcher the trouble of manually labelling the supervised data. Bengio et al. (2003) [117] and Collobert et al. (2011) [118] were the first to use self-supervision to perform the task of neural language modelling, where the next word of a running text is used as a supervision signal.

Word prediction is usually performed by applying the softmax operation on the outputs of the neural network and computing probabilities of each word in the vocabulary being the golden answer. An obvious problem with the softmax operation is that it is computationally expensive (of $O(|V|)$ complexity). Mikolov et al. (2013) [116] solved this problem by replacing word prediction with binary classification. SGNS uses negative samples, i.e. for each and every word found in the context of the current word they sample k randomly chosen words that are not found in its context and perform $k + 1$ binary classification tasks asking the model whether each of the $k + 1$ words is found in the current word's context or not. For example, in the case of the phrase "$\cdots$ was looking at [the cloudy sky I noticed] two $\cdots$" the current word is "sky" and the the four context words in this 5-word window are "the", "cloudy", "I" and "noticed". For each of the context words k random words not found in the context are sampled. The negative samples are sampled according

to the weighted unigram frequency:

$$p_a(w) = \frac{count(w)^a}{\sum_{w'} count(w')^a} \tag{3.10}$$

where a was chosen to be equal to 0.75 in the paper.

Their model is very simple. The probability of a word c being in the context of the current word w is modelled as:

$$p(+|w,c) = \sigma(\mathbf{c}\mathbf{w}^T) = \frac{1}{1+exp(-\mathbf{c}\mathbf{w}^T)} \tag{3.11}$$

It follows that:

$$p(-|w,c) = 1 - p(+|w,c) = \sigma(-\mathbf{c}\mathbf{w}^T) = \frac{1}{1+exp(\mathbf{c}\mathbf{w}^T)} \tag{3.12}$$

The goal is to maximize the probability of each positive sample being in the current word's context while minimizing the same probabilities for its k negative samples:

$$\begin{aligned} L_{CE} &= -\log[p(+|w,c_{pos}) \prod_{i=1}^{k} p(-|w,c_{neg,i})] \\ &= -[\log \sigma(\mathbf{c}_{pos}\mathbf{w}^T) + \sum_{i=1}^{k} \log \sigma(-\mathbf{c}_{neg,i}\mathbf{w}^T)] \end{aligned} \tag{3.13}$$

The weights are trained with gradient descent and for each word two vectors are trained, a target embedding, $\mathbf{w}$, and a context embedding, $\mathbf{c}$. It is both possible to use their sum or discard $\mathbf{c}$ and just use $\mathbf{w}$ as the final embedding.

Many other methods to create dense embeddings have since been suggested. **fasttext** was proposed by Bojanowski et al. (2017) [119] and deals with the problem of unknown words present in SGNS. **GloVe** (Pennington et al. (2014) [120]) is based on capturing corpus global statistics by computing ratios of probabilities from a word-word co-occurrence matrix [9].

3.4.3 Embedding Similarity

To measure word similarity with the use of word embeddings the **cosine similarity** is usually employed. Cosine similarity measures the cosine of the angle between two words' embeddings:

$$sim(w_i, w_j) = \frac{\mathbf{w}_i \mathbf{w}_j}{||\mathbf{w}_i|| ||\mathbf{w}_j||}, \text{ where } \mathbf{w}_i, \mathbf{w}_j \text{ are the two embeddings} \tag{3.14}$$

The bigger the metric the more similar the two words are considered to be [9].

3.4.4 Evaluating Vector Models

Like language models, vector models can also be evaluated extrinsically or intrinsically. The two most common tasks used for intrinsically evaluating vector model are the similarity task and the analogy task. Datasets of **similarity tasks** are created by having humans assign similarity scores to pairs of words with (SCWS, Huang et al. (2012) [121] and WiC, Pilehvar and Camacho-Collados (2019) [122]) or without (WordSim-353, Finkelstein et al. (2002) [123] and SimLex-999, Hill et al. (2015) [124]) the use of context.

An **analogy** is of the form a is to b what c is to d and is symbolized as $a : b :: c : d$. The task is to find d, given a, b and c (Turney and Littman (2005) [125]). Analogy datasets contain sets of four words that satisfy an analogy relationship (Mikolov et al. (2013a) [116], Mikolov et al. (2013b) [126], Gladkova et al. (2016) [127]). Examples may use different word morphology, e.g. *give* : *giving* :: *take* : *taking*, lexicographic relations, e.g. *wheel* : *car* :: *trunk* : *tree*, or encyclopedic

knowledge, e.g. *Alps* : *Europe* :: *Himalayas* : *Asia*. Mikolov et al. (2013a) [116] used simple vector addition and subtraction to perform the analogy task, e.g. $Paris - France + Italy = Rome$ and reported good results.

3.5 Neural Language Models

n-gram language models have two major weaknesses. First, they are unable to generalize when presented with words that are different from the ones they have seen during training, even though they may be related to each other. Take for example, the training set sentence "I love the blue colour of the sea" and a possible test sentence "I Love the blue colour of the ocean". A n-gram model that has never seen the second sentence in the training set will assign a zero probability to the word "ocean" following the phrase "I love the blue colour of the". But a neural model that is trained in the same training set knows that the words "sea" and "ocean" have similar meanings and will assign a nonzero probability value to the second phrase.

Second. n-gram models use limited context, which means that if information that can be used to accurately predict a certain word appears in the text more than n tokens before this word the n-gram model will miss it. This is the case for example in sentence "I want to go surfing. Could you Jerry ask Jim and Jenny if they would like to come with us when we go to the sea?". A 5-gram model will not use the word "surfing" and thus won't score the word "sea" as high as it should.

3.5.1 A Feed-Forward Neural Network As A Language Model

Neural networks are usually employed to solve these problems. A LM could be implemented by a FFNN that will take as input at time t the embeddings of C words $(w_{t-C}, \cdots, w_{t-1})$ and use the softmax operation to output a probability distribution over the set of possible next words. A simple FFNN architecture performing this task for $C = 3$ could be:

$$\mathbf{e} = [\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}] \tag{3.15a}$$

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{e} + \mathbf{b}) \tag{3.15b}$$

$$\mathbf{z} = \mathbf{U}\mathbf{h} \tag{3.15c}$$

$$\hat{\mathbf{y}} = softmax(\mathbf{z}) \tag{3.15d}$$

where $\mathbf{E}$ is the embedding matrix of dimensions $d \times |V|$ and $\mathbf{x}_i$ is a one-hot vector with an ace in the position that corresponds to the respective token. y is the true label given in a one-hot vector form with $|V|$ elements. For a single example the loss function can be written as:

$$L_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{n=1}^{|V|} y_n \log \hat{y}_n = -\sum_{n=1}^{|V|} \mathbf{1}\{y_n = 1\} \log \hat{y}_n = -\log \hat{y}_w \tag{3.16}$$

where w is the golden token and $\mathbf{1}\{y_n = 1\}$ is only equal to 1 when $n = w$.

Note that the embeddings need not be learned simultaneously with the rest of the model's parameters. Instead they can be learned before the model is trained with the use of methods similar to the ones described in 3.4.2. This matter will be extensively discussed in 3.9.2. Moreover, sometimes the embedding layer may be further trained along with the rest of the model parameters or may be frozen to prevent the initial embeddings from losing valuable information.

This model presented here is able to generalize to related words and contexts but does not solve the limited context problem, since it can only see C steps in the past. Also it makes it difficult to learn patterns that are independent of the position in the sequence. The phrase "but not"

for example needs not be processed differently when it is found in the 2nd and 3rd positions and differently when it is found in the 1st and 2nd positions one step later.

3.5.2 Recurrent Neural Network As A Language Model

RNNs solve the limited context problem since relevant information is stored in the context vector. Thanks to the weight sharing prior across different positions RNNs also solve the second problem. A RNN based on the model presented in chapter 2.6.1 can be described by the following equations:

$$\mathbf{e}_t = \mathbf{E}\mathbf{x}_t \tag{3.17a}$$

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \tag{3.17b}$$

$$\hat{\mathbf{y}}_t = softmax(\mathbf{V}\mathbf{h}_t) \tag{3.17c}$$

They can be trained with self-supervision described in 3.4.2, i.e. by using a corpus of text and forcing the model to predict the next token while moving the context window one position to the right at each step. The model is trained to minimize equation 3.16 at each step and a teacher forcing training technique is followed (chapter 2.6.2), i.e. the input to the model is the golden token sequence irrespective of its previous predictions. This processed is shown in figure 3.1. The goal is to minimize the average loss and GD can be used [9].

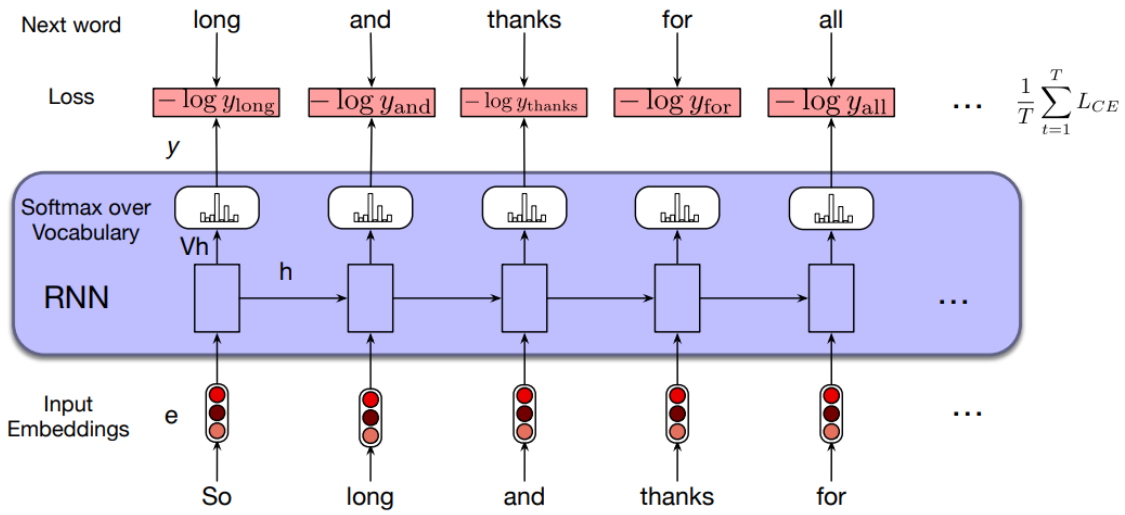


Figure 3.1. *Example of a recurrent neural network being used for language modeling. At each step it receives as input the correct previous token and the hidden state created during the previous step and generates a probability distribution over the vocabulary tokens using a softmax layer [9]*

3.5.3 Recurrent Neural Networks For Other NLP Tasks

RNNs are thus a natural model selection for NLP tasks. With minor modifications they can used in a wide range of such tasks:

- sequence labelling: e.g. pos tagging and NER, where a probability distribution over tags is generated at each step using a softmax layer
- sequence classification: e.g. spam detection and sentiment classification, where a summary of the input sequence is created by a RNN and used by a FFNN followed by a softmax layer that outputs a distribution over the possible class labels

- generation: e.g. question-answering (QA), summarization and dialogue. LM implemented by RNNs can be used as auto-regressive models as noted in chapter 2.6.5.

The methods and models that will be discussed in the following sections were first tested on neural machine translation problems. That is why it is beneficial to carefully examine this problem.

3.6 Neural Machine Translation with RNNs

Neural machine translation models are trained on **parallel corpora**, i.e. documents that are translated in two or more languages. **Neural machine translation (NMT)** can be probabilistically interpreted as trying to maximize the conditional probability of the output sentence y given a source sentence x: $argmax_y p(y|x)$. The theoretical training goal thus is to maximize the conditional probability of each of the sentence pairs that make up the training corpus. Once this is achieved the model can hopefully generate the most probable translation for any given source sequence.

A sentence is thus represented as a sequence of vectors, where each vector is the embedding of the word in the corresponding sentence position. In NMT, the neural models receive sequences of word embeddings corresponding to sentences that are written in a source language as input and are expected to output their translation in the target language.

Since the two sentences might have different lengths a RNN encoder-decoder model is used. The encoder RNN processes the input sentence and creates a context vector that summarizes it. The decoder RNN produces a translation in a sequential fashion by receiving at every step the embedding of the previous word it generated, while also being constantly conditioned on the context vector produced by the encoder.

From a probabilistic perspective a decoder decomposes the joint probability into conditional probabilities as follows:

$$p(y) = \prod_{t=1}^{T_{out}} p(y_t|\{y_1, \cdots, y_{t-1}\}, \mathbf{c})$$

where c is the context vector. An RNN decoder, that only uses the previously generated word as input, models each conditional probability as:

$$p_{seqtoseq}(y_t|\{y_1, \cdots, y_{t-1}\}, \mathbf{c}) = g(y_{t-1}, \mathbf{h}_{d,t}, \mathbf{c})$$

where g is a nonlinear function and $\mathbf{h}_{d,t}$ is the decoder's hidden state at the t-th step.

The embedding space of the output vocabulary is a different one. This means that it is possible for two words that belong in two different vocabularies to have the same embedding. But, it also means that it is not even necessary for the respective embedding spaces to be of the same dimension.

Evaluating Machine Translation Systems

Evaluating machine translation systems is not as easy as evaluating models that solve pos tagging, that can be viewed as a multi-class classification problem. A produced translation may use different words from the reference one but still be a valid translation of the source sentence.

Therefore it is usual to have humans evaluate model-produced translations. Bilingual raters can read both the source sentence and the generated translation and then evaluate the latter. If bilingual raters are not available, monolingual raters can alternatively be given good human translations of the source sentences, which they will then compare to the generated ones.

Having humans rate translations is slow and inefficient. That is why automatic evaluation methods are often employed. A family of automatic evaluation methods uses the character overlap between the human translation and the output of the model. As first observed by Miller and Beebe-Center (1956) [128], machine translations of high quality tend to use the same words and characters that exist in the reference translation. A well known technique that belongs in this family of methods is **chr F** (Popovic (2015) [129]), which uses the n-gram overlap of the generated translation with the reference translation to rank the first one. After the specification of the hyper-parameter n, chrF calculates the percentage of unigrams, bigrams, $\cdots$ and n-grams in the reference translation that also exist in the generated translation as well as the opposite. The first ones are the precisions, symbolized as chrP, and the second ones the recalls, symbolized as chrR. chrF is then computed as:

$$chrF_\beta = (1 + \beta^2)\frac{chrP \cdot chrR}{\beta^2 chrP + chrR} \tag{3.18}$$

An alternative overlap metric is **BLEU** (BiLingual Evaluation Understudy, Papineni et al. (2002)), which only uses precision-based quantities. Specifically, it combines n-gram word precision with a brevity penalty.

A problem with methods based on character overlaps is that they penalize small variations from the reference translation and the use of different words which, however, may not significantly alter the generated sentence's meaning. Evaluation methods that use **embeddings** on the other hand are able to overlook these minor differences. It is usual for such models to match contextual embeddings generated by BERT-like models, chapter 3.9, for the reference translation and the model output, and then use the matching degree as an evaluation metric. An example is the $\text{BERT}_{\text{SCORE}}$ algorithm (Zhang et al. (2020)) [130]. This is based on the intuition that embeddings of phrases with similar meanings will be close in the vector space. On the other hand, suppose that a dataset that consists of reference and machine-generated translations of the same source sentences, but also contains the respective human ratings for the generated translations, is available. Then it is possible to build neural models that, given the two sentences, can predict a rating for a generated translation. Such models can then be used to automatically rank a generated translation (COMET, Rei et al. (2020) [131], BLEURT, Sellam et al. (2020) [132]).

3.7 Attention

The performance of vanilla RNN models - the simplest possible version of model is called a vanilla version - is known to degrade when the length of the input sequences is increased due to the vanishing gradients problem. This is also the case with RNN type encoder-decoder models that were discussed in chapter 2.6.5. Even worse, in the encoder-decoder case information about the entire input sequence has to be encoded in a fixed-size context vector, a problem known as **information cramming**.

3.7.1 Encoder-Decoder With Attention

Bahdanau et al. (2014) [10] proposed the **attention mechanism** in an effort to solve the aforementioned problem and tested their model on NMT tasks. The attention mechanism provides the decoder with information of all hidden states created by the encoder while it was processing the input. It weighs the hidden states based on their relevance in every decoding step and computes their weighted sum. The decoder's processes are then conditioned on this vector to produce the next word. Since all hidden states are simultaneously available to the attention mechanism, attending to distant words now becomes possible and as easy as attending to nearby ones.

Model

Instead of $g(y_{t-1}, \mathbf{h}_{d,t}, \mathbf{c})$, $g(y_{t-1}, \mathbf{h}_{d,t}, \mathbf{c}_i)$ is defined to be the new function implemented by the decoder. The vector $\mathbf{c}_i$ is different for each output word, indicating that it is computed to specifically assist the decoder in choosing the next word. The context vector $\mathbf{c}_i$ is defined as follows:

$$\mathbf{c}_i = \sum_{j=1}^{T} m_{ij} h_{e,j}, \text{ where } m_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_{in}} exp(e_{ik})} \tag{3.19}$$

where $e_{ij} = a(h_{d,i-1}, h_{e,j})$ is an alignment model that is implemented by a ffnn and is trained jointly with the rest of the model. It is responsible for scoring the relevance of each hidden state of the encoder to the current hidden state of the decoder. When the decoder attempts to predict the i-th output word, it is only reasonable for the alignment model to pay more attention to hidden states of the encoder that correspond to the parts of the input sequence that are most useful for predicting y_i. Their respective scores, e_{ij}, will thus be larger reflecting their relevance. Since all of the encoder's hidden states are available to the decoder, the encoder does not need to summarize the entire input sentence into a single vector; it only has to produce informative hidden states for every part of the input.

For the encoder model, Bahdanau et al. (2014) [10] used a BiRNN to obtain information from both previous and following words.

3.7.2 Decoding Using Beam-Search

During training the model tries to maximize the probability of choosing the correct words. Thus, during inference, a possible decoding strategy would be to choose at every step the token that is deemed by the model to be the most probable one of being next. The problem with this **greedy decoding method** is that choosing the token to which the model the larger probability value is not always the best choice, and might even turn out to negatively affect the translation quality later. Ideally, one would use **tree search**, extending the leaves at every decoding step and re-computing the probability of each path as such: $p_{path} := p_{path} \times p_{token}$. When the decoder has generated a end_of_seq token for each path or when the maximum tree height has been achieved the decoding stops. The chosen sequence is the one with the highest cumulative probability.

Tree search would indeed find the most fitting token sequence, but its complexity is exponential w.r.t. the length of the output sequence since each leaf is extended using the entire vocabulary. Instead, a computationally cheap alternative is usually employed, called **beam-search**. Beam-search retains k possible token sequences at each time step. Using each one of them as input it computes k distinct probability distributions over the vocabulary tokens using the softmax layer. This leads to $k \times |V|$ hypotheses. It then chooses the k top ranking ones and moves on to the next step. The sequence with the largest cumulative probability is finally chosen. An example is presented in figure 3.2 [9].

Results

Bahdanau et al. (2014) [10] used beam-search decoding. Their model outperformed vanilla RNN encoder-decoder models, especially in the translation of long sentences. Figure 3.3 shows how the model attends to the input sequence to produce a translation. Hidden states of RNNs are known to be most strongly related to inputs that have been most recently processed by the model, i.e. $\mathbf{h}_{e,j}$ will most likely focus on $\mathbf{x}_j$ and the inputs near it. This is evidently shown in the way the weights for the encoder's hidden states are chosen.

The attention mechanism introduced there inspired the Transformer model that is the basis

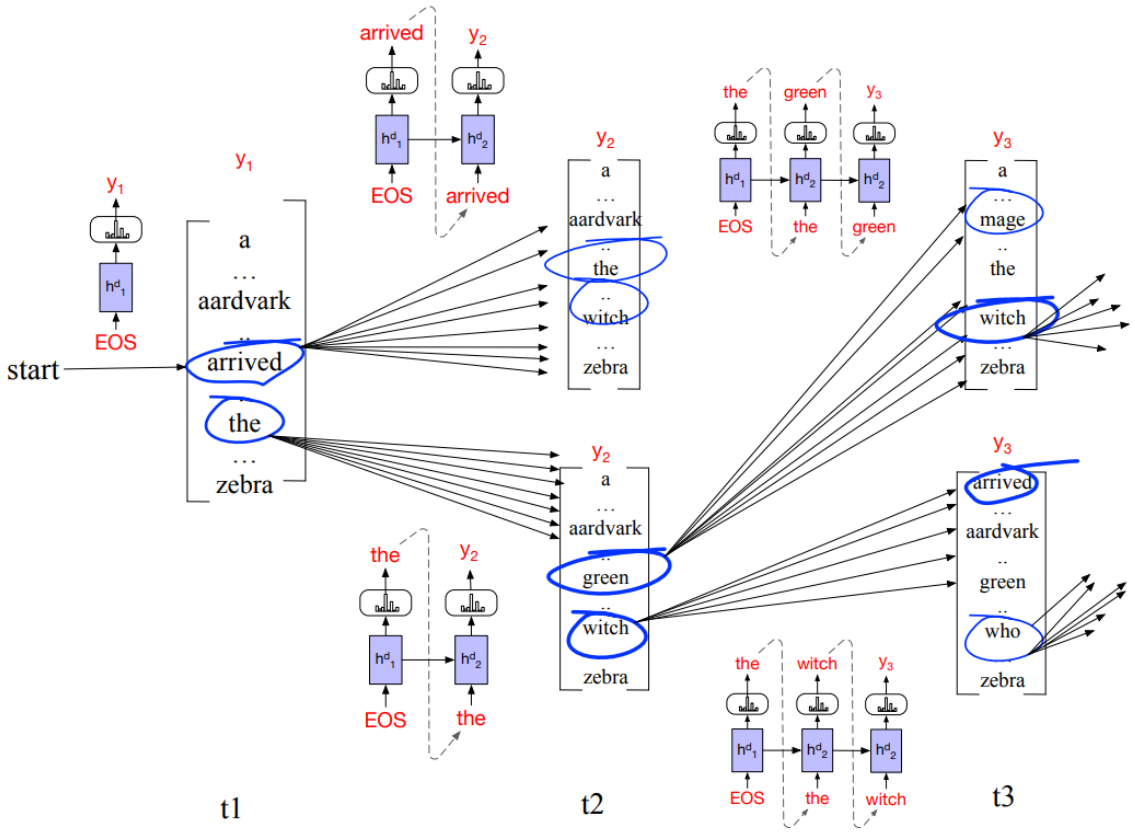


Figure 3.2. *Example of beam search for k=2. In the beginning the most probable starting words are chosen from a single distribution. At each next step 2 new distributions are computed, 1 for every active path. Then the best 2 paths are then chosen. Notice that in t2 it is the case that both paths result from the same leaf, while in t3 each leaf is extended once [9]*

of most modern state-of-the-art NLP models. What is more is that the attention mechanism is another example of a successful incorporation of prior knowledge into a model's architecture (chapter 2.5.2). This matter will be further analyzed in chapter 4.

3.8 Transformer

The attention mechanism mitigated the vanishing gradients problem but, due to the sequential nature of recurrent neural models, the computations cannot be parallelized. Parallelization is incredibly important for speeding up training, and thus allowing the researcher to benefit from data abundance and create bigger models with increased expressive power.

Vaswani et al. (2017) [1] created the **Transformer**, an encoder-decoder neural model that employs the attention mechanism and the weight sharing prior to allow for sufficient parallelization of the computations.

3.8.1 Attention in the Transformer Model

Similarly to Bahdanau et al. (2014) [10] they tested their model on the task of NMT. They view attention as performing a mapping, whose result depends on the compatibility of a query vector with several key vectors, each one corresponding to a value vector. More specifically, the level of compatibility of each key vector with the query vector determines the contribution of the respective value vector to the mapping. In the case of an RNN-type encoder-decoder model with

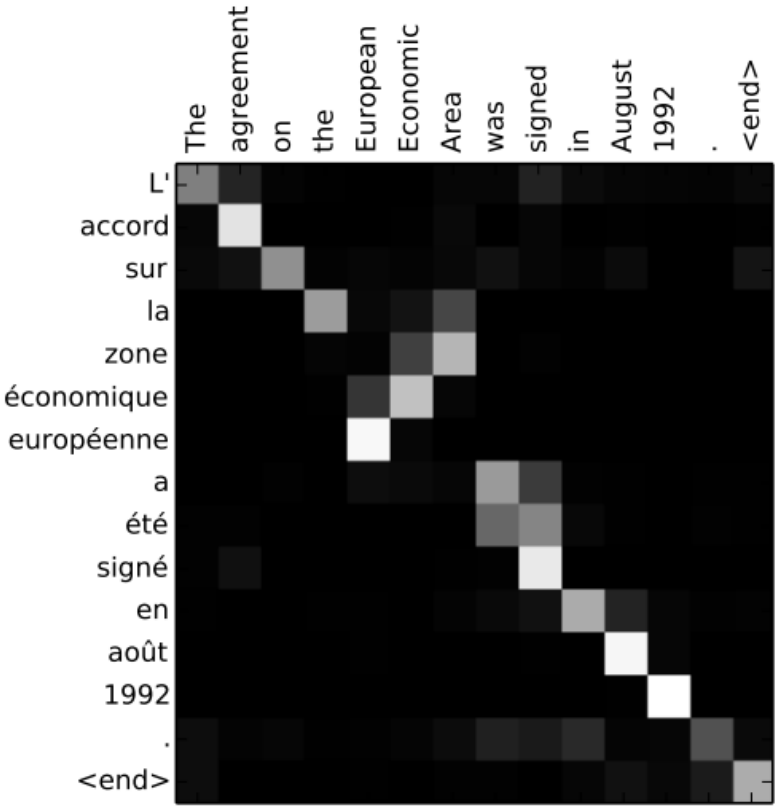


Figure 3.3. *Example of attention weights. The horizontal axis corresponds to the source sentence (English) and the vertical axis to the generated translation (French). Every pixel shows an alignment weight m_{ij}, where $\sum_{j=1}^{14} m_{ij} = 1$ [10]*

attention, as described by equation 3.19, the query is the decoder's hidden state and both the key and value vectors are the hidden states of the encoder. Vaswani et al. (2017) [1] perform multiple attention operations in parallel for a variety of query vectors. They store queries, keys and values in the matrices $\mathbf{Q}_{max\_seq\_len \times D_{key}}$, $\mathbf{K}_{max\_seq\_len \times D_{key}}$ and $\mathbf{V}_{max\_seq\_len \times D_{val}}$ respectively. Moreover Bahdanau et al. (2014) use a separate ffnn to compute attention weights. In contrast to that, Vaswani et al. (2017) [1] use **Scaled Dot-Product Attention** which computes the query-key compatibility values by computing their dot product, hence the use of vectors of the same dimension. Scaled Dot-Product Attention is performed as follows:

$$Atten(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D_{key}}})\mathbf{V} \tag{3.20}$$

They divide the dot products by the dimension's root in order to decrease the input values to the softmax, which tend to grow large as vector dimensions increase and enter regions where the softmax's gradients become extremely small.

They extensively use a variant attention called **self-attention** that relates positions of the same sequence in order to produce contextualized representations, i.e. the three matrices come from the same sequence. In the case of written language, self-attention updates the words' embeddings by injecting information coming from their context. These embeddings are often called **contextual embeddings**.

They also use the encoder-decoder attention variant described previously with the exception, of course, of using dot products instead of an alignment model. Moreover, they employ **multi-**

head attention to increase the flexibility of the attention mechanism. Instead of using the element representations themselves, they linearly project them to vector spaces of dimension h times smaller than the initial ones, where h is the total number of heads. They then perform h different attention operations in parallel, each one with a distinct triplet of $\mathbf{q}$, $\mathbf{k}$, $\mathbf{v}$ vectors. This allows every head to pay attention to a different set of features of the input, promoting its specialization. The h outputs are then concatenated and again projected.

$$\begin{aligned} MultiHead(\mathbf{Q},\mathbf{K},\mathbf{V}) &= Concat(head_1,\cdots,head_n)\mathbf{W}^O, \\ \text{where } head_i &= Attn(\mathbf{Q}\mathbf{W}_i^Q,\mathbf{K}\mathbf{W}_i^K,\mathbf{V}\mathbf{W}_i^V), \\ \mathbf{W}_i^Q \in \mathbb{R}^{D_{model}\times D_{key}}, \mathbf{W}_i^K &\in \mathbb{R}^{D_{model}\times D_{key}}, \mathbf{W}_i^V \in \mathbb{R}^{D_{model}\times D_{val}}, \mathbf{W}^O \in \mathbb{R}^{hD_{val}\times D_{model}} \end{aligned} \tag{3.21}$$

3.8.2 The Model

Encoder

The transformer model consists of an encoder and an encoder and a decoder. The encoder is a stack of $L = 6$ layers, each of which is made of the same sub-layer types. The first sub-layer of every layer is a multi-head self-attention mechanism. Each position attends to the outputs of all positions of the previous layer. The second sub-layer is a fully connected ffnn that implements a ReLU function:

$$ffnn(x) = max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2,\ \mathbf{W}_1 \in \mathbb{R}^{D_{model}\times D_{ffnn}},\ \mathbf{W}_2 \in \mathbb{R}^{D_{ffnn}\times D_{model}} \tag{3.22}$$

and is applied separately to each and every position. The parameters of the two sub-layers of the same type that belong to different layers are distinct. But weight sharing is employed for neural modules operating on different positions of the same layer, in the same spirit that deep RNNs use different weights across neural layers but the same weights to process each sequence element. Around each sub-layer a residual connection is applied to allow free flow of gradients and the sum is normalized by the layer normalization technique, chapter 2.4.5.

Decoder

The decoder is similar to the encoder but with two important differences. First, a third sub-layer is placed in between the other two in each layer. This sub-layer implements an encoder-decoder multi-head attention mechanism, extracting, for each position, queries from the previous sub-layer and key and values from the positions of the encoder's final output. Its role is similar to the role of the attention mechanism studied in the previous chapter, 3.7.1, that enables the decoder to attend all positions of the input after they are processed by the encoder.

Second, a constraint is imposed to allow the decoder's training to be parallelized. In order for that to happen, the entire target sentence must be provided to the decoder and used in a teacher forcing manner, where the decoder simultaneously attempts to predict all target words, while being allowed to use for the generation of each word only words that precede it in the target sequence. A problem is thus created, since all positions of the decoder are simultaneously active trying to predict the next words they have access, through the self-attention mechanism, to the next word that they are trying to predict. Vaswani et al. (2017) [1] solve this problem by applying a triangular mask over self-attention weights, forcing each individual self-attention mechanism to pay attention only to outputs corresponding to previous positions and to the output of its own position. An example of the application of the triangular mask is shown in figure 3.4.

On top of each separate decoder's position a learned linear transformation and a softmax layer are applied in order to compute probabilities for the possible next words.

Figure 3.4. *Example of an application of a triangular mask in the decoder self-attention. The scores are masked before a softmax layer is applied to transform them into weights. Notice how the first token is only allowed to attend to itself. Figure from* ***https://jalammar.github.io/illustrated-gpt2***

The model's outline is shown in figure 3.5.

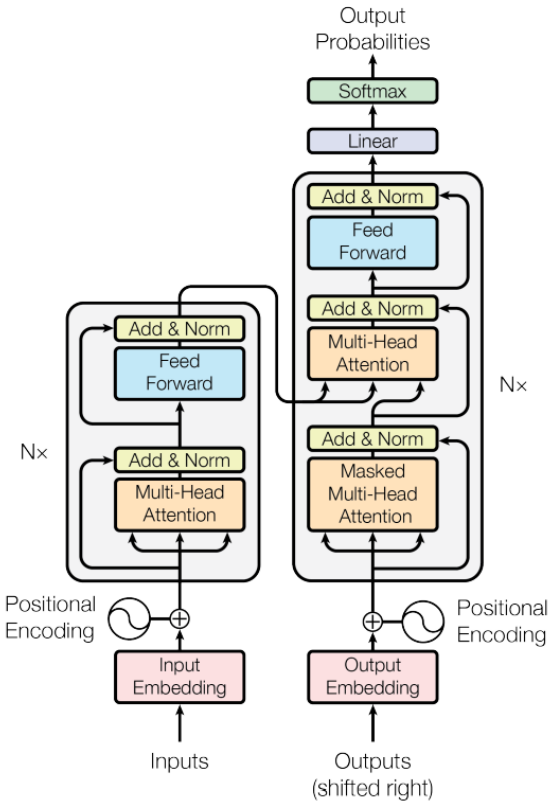


Figure 3.5. *Transformer model outline [1]*

Embeddings and Dropout

They use learned embeddings of dimension D_{model} for both languages. But, in contrast to RNN models, and since weights are shared across positions, there is no information to indicate the positions of the words in the input sequences of each model. Therefore, they generate **positional embeddings** according to sinusoids whose frequencies depend in a predictable manner on the position of the embedding in the input sequence. These are added to the word embeddings to inject position-related information.

Finally they apply dropout to the outputs of the sub-layers and to the sum of word and positional embeddings.

3.8.3 Results

Transformer outperformed sota models of its time on a variety of machine translation datasets, while requiring less training time. Vaswani et al. (2017) [1] included the table 3.1 shown below regarding the benefits of using self-attention instead of recurrence or convolution to create contextual embeddings.

The methods are compared w.r.t.:

- the total computational complexity per layer, showing that self-attention's complexity is linear w.r.t. the representation dimension in contrast to the cases of recurrence and convolution. This allows the use of bigger embeddings increasing their expressive power. Notice that self-attention is quadratic w.r.t. the sequence length due to the matrix multiplications, but h is usually chosen to be smaller that d.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Length Path
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d)$	$O(1)$	$O(\log_k n)$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Table 3.1. *Table comparing several layer types w.r.t. three different attributes. n is the length of the input sequence, d is the dimension of the vectors representing each sequence's element, k is the size of the kernels of a convolutional layer and r is the number of elements that a self-attention operation is allowed to pay attention to*

- the number of sequential operations indicating a great disadvantage of RNNs that hinders parallelization.
- the maximum path length between sequence elements that determines the difficulty of learning long-range relationships. Self-attention is again the winner since a query-key compatibility check is performed for all available keys.

3.9 Neural Model Pre-training

3.9.1 Contextual Embeddings

The encoders presented in the previous chapters were responsible for creating representations of the input on which decoding was then conditioned. It is natural for someone to wonder what if, instead of performing attention on an encoder's outputs, the initial input sequence embeddings were used without passing through a neural module. Even though that is entirely possible, it has been known to yield inferior results compared to the use of an encoder-decoder neural model. As was noted in chapter 3.4, words derive their meanings from the contexts in which they are found. But, even though a word can be **polysemous**, e.g. have multiple meanings, that are expressed in different contexts, it must be represented by a single embedding. Even though an instance of the word "mouse" can be used to describe an animal, while another one to refer to a part of a computer, an embedding creating algorithm like the one described in the section 3.4.2 will not process the two instances separately. The same thing is true for words whose meanings change only slightly in different contexts, like the word "glass" in the phrases "window glass" and "a glass of water". Their embeddings will necessary result from a combination of all these meanings in one, Arora et al. (2016) [133]. It would thus be very possible for a decoder that would use the word embeddings without any pre-processing to become confused as to how some input words are used.

Encoders create what is known as **contextual embeddings**, i.e. embeddings that capture the way words are used in the contexts in which they are found. An encoder is in charge of deciding in which way the word "mouse" is used in a particular phrase and provide the decoder with a fitting embedding so that it can then focus exclusively on the task of word generation. In other words, the encoder deals with the ambiguity of language making the life of the decoder easier.

3.9.2 Pre-training and Fine-Tuning Neural Models

Learning representations for a natural language's words and phrases in way that allows its understanding, interpretation and generation is an extremely data-intensive task. The available data for some tasks, such as biomedical ones, does not suffice for the model to effectively learn these representations. It has been found that, instead of learning word embeddings from scratch, it is usually beneficial to initialize the word embeddings of a model with learned ones, trained

via language modelling tasks on independent very large corpora. The same is known to be true with neural encoders, that are in charge of creating contextualized embeddings. After parameter initialization, these models can be trained on large unlabeled, and thus relatively cheap to acquire, corpora. This training stage is called **pre-training**.

One can then use these trained models to perform a downstream task. This is done by adding on top of them small, randomly initialized neural modules that are trained jointly with the pre-trained model on the new task's training data. The pre-training stage is usually performed using self-supervision, while this phase usually makes use of manually labeled samples. But, because the model is already familiar with natural language it only needs to learn task-specific knowledge, which dramatically reduces the number of samples that are needed for this to happen. This process is known as **fine-tuning**. Pre-training and then fine-tuning is a paradigm of what is known as **transfer learning**, i.e. the process of employing the knowledge of dealing with a task in order to handle a new task.

The structure of the model depends on the type of downstream tasks it is destined to undertake. In the case of a Transformer-like encoder the following modifications are possible:

- for sequence classification tasks like emotion detection, an additional vector $[CLS]$ can be added to the input sequence and then processed by the encoder just like the rest. The corresponding output is trained during both pre-training and fine-tuning to contain information about the entire input sequence in a way that assists the classification task. A structure as simple as a single-layer FFNN with a softmax layer on top can be used to process the output vector and perform the classification. The task-specific structure can be trained along with the rest of the model during fine-tuning.

- for pair-wise sequence classification tasks like natural language inference (NLI), the same vector $[CLS]$ can be used. The pair of sentences is separated by a $[SEP]$ token and the output of the $[CLS]$ token is used as a summary of the task-relevant information that will assist the classification.

- for sequence labelling tasks like pos tagging, the contextual embeddings are taken into consideration. The task-specific structure is again a classifier implemented by a FFNN followed by a softmax layer that uses each of the aforementioned vectors as input to label the corresponding token. Signals from all the classifications are propagated backwards during the fine-tuning of the model.

- for span-based applications like NER and QA, the problem can be formulated as trying to maximize $p(a|q,p)$, where $q = q_1 \cdots q_{L_q}$ is the question, $p = p_1 \cdots p_{L_p}$ is the passage and a is the possible answer spans. This probability can be simplified to

$$p(a|q,p) = p_{start}(a_s|q,p)p_{end}(a_e|q,p) \tag{3.23}$$

  which translates into finding the token that is most likely to be the starting point of the answer as well as the token that is most likely to last one of the span, under the constraint that $s \leq e$.

  Each contextual embedding is used as input to two FFNNs, one that computes $p_{start}(a_s|q,p)$ and another one that computes $p_{end}(a_e|q,p)$. The training loss is equal to:

$$L = -\log p_{start}(a_i|q,p) - \log p_{end}(a_j|q,p) \tag{3.24}$$

  where i and j are the correct starting and end points of the span respectively. Moreover the output corresponding to a $[CLS]$ can also be used as input to another estimator that

estimates the probability that a passage that satisfies the requirements does not exist in the input sequence [9].

3.9.3 Pre-Trained Deep Bidirectional Transformers

Prior Work

The appearance of models create contextualized embeddings only increased with publication of the Transformer paper. Radford et al. (2018) [134] proposed **GPT** (Generative Pre-Training), a Transformer decoder pre-trained on LM tasks. Since the model is unidirectional, information is propagated from left to the right. Part of the information that is present in the input sequence is thus not available to the model when it computes the contextualized representations, which limits the capabilities of the model.

Peters et al. (2018) [135] proposed **ELMo** (Embeddings from Language Models) representations. ELMo result from the concatenation of the representations generated by two unidirectional LSTMs that traverse the input sequence heading towards opposite directions, one from right to left and the other from left to right. Devlin et al. (2018) [11] refer to ELMo as a shallow bidirectional model and contend that its potential is limited because of the naive way in which the bidirectional information is processed.

Pre-Training Tasks

They propose the use of **BERT** (Bidirectional Encoder Representations from Transformers) that are acquired by allowing self-attention layers to use information from both sides of the sequence, like the self-attention layers used by the encoder of the Transformer layer. The most significant innovation proposed by Devlin et al. (2018) [11] is the use of **masked language modeling (MLM)** technique inspired by the Cloze Task (Taylor, 1957) [136]. The problem with using a Transformer encoder to perform language modelling is that the model is given the target word as input rendering the task trivial. In order to avoid this, Devlin et al. (2018) [11] mask the input token of the target word and then use the generated output of the respective position to predict it. The encoder has to infer the target word from its context.

In fact, the algorithm randomly chooses 15% of the input tokens to be masked and then predicted, while allowing the use of the rest as context. Moreover, to avoid a mismatch between training and inference conditions due to the use of masks during training but not during inference, they randomly choose to leave the original token unmasked in 10% of the target words and choose to replace it with another randomly selected token in another 10% of the cases. Since BERT does not known which are the target words it is forced to maintain information about all input tokens.

In order for the model to be taught how to handle sentence pairs they also deploy a **next sentence prediction (NSP)** task. They sample pairs of sentences from the training corpus out of which 50% are consecutive sentences ad 50% are not. They then ask the model to predict whether the second sentence is the one that follows the first one in the corpus or not. The two sentences are separated by a $[SEP]$ token and two learned sentence embeddings, A and B are added to every word of each sentence respectively. Since the two sentences are concatenated, one self-attention mechanism performs in essence bidirectional cross-attention between them.

They also use pre-trained word embeddings and add positional embeddings apart from the sentence ones. In the case of a single sentence they only add sentence A embeddings.

The pre-training setup is shown in figure 3.6.

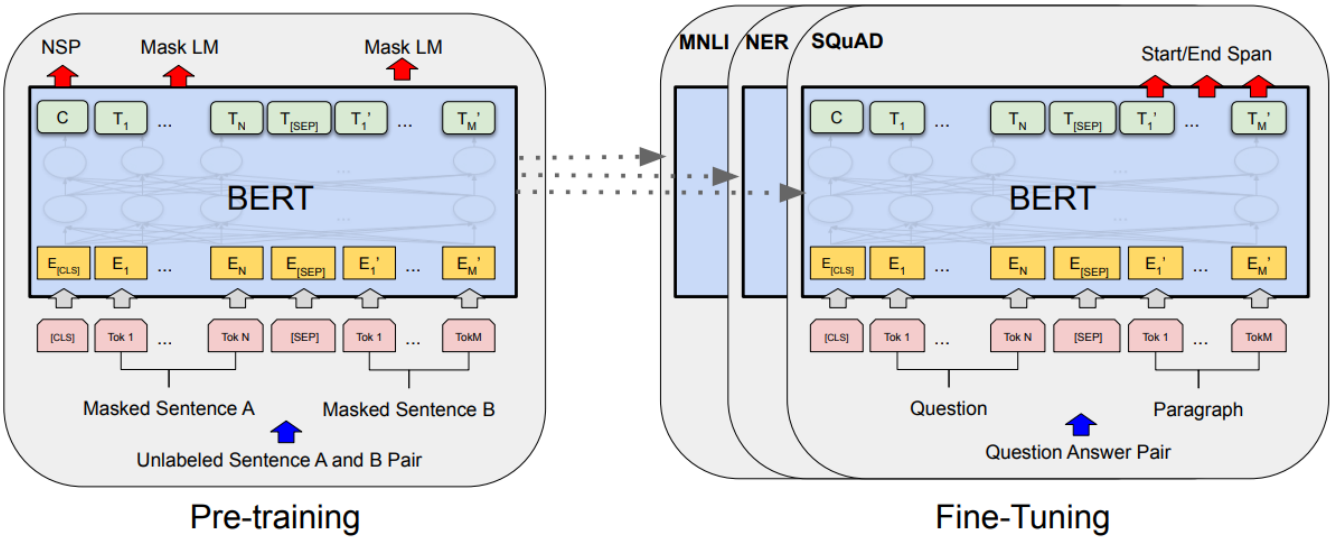


Figure 3.6. *BERT setups for the pre-training and fine-tuning procedures. As far as pre-training is concerned the use of the output of the [CLS] vector for the NSP task and of token vectors for the MLM task are depicted. The use of token outputs for QA downstream tasks is also shown [11]*

Pre-Training Corpora And Models

BERT is pre-trained on a BookCorpus [137], a 800 million word corpus of book texts that is currently not available for reasons of intellectual property. It is also trained on a 2.5 billion word corpus form the English Wikipedia. It must be noted that newer models are usually pre-trained on much larger corpora.

Devlin et al. (2018) [11] pre-trained a base model with 110 million parameters and large one with 340 million parameters.

Fine-Tuning Tasks And Models

During fine-tuning all model parameters are fine-tuned and task-specific neural modules are added on top to perform the downstream tasks. The model is fine-tuned to GLUE benchmark tasks. The **General Language Understanding Evaluation (GLUE)** benchmark is a set of tools created by Wang et al. (2018) [138] in order to provide a holistic approach to NLP model evaluation. The authors of the paper gathered a variety of NLP tasks and performed certain modification wherever they deemed was necessary to facilitate the training and evaluation of neural models at each one of them. Due to the data scarcity in some of the tasks, training a model at each one of them separately without pre-training it first does not yield competitive results. Models must therefore already possess knowledge of the English language before they specialize in each of the benchmark's tasks. The benchmark consists of the following single-sentence tasks:

- CoLA (Corpus of Linguistic Acceptability, Warstadt et al. (2018) [139]): its corpus contains word sequences drawn from books and articles on linguistic theory, that are annotated with whether they are grammatically correct or not
- SST-2 (Stanford Sentiment Treebank, Socher et al. (2013) [140]): its corpus is made of movie reviews annotated based on their sentiment

As shown in figure 3.7(b) and explained in chapter 3.9.2 the output of the $[CLS]$ token can be used as input to a FFNN that performs the classification. As depicted in figure 3.7(a) the output of the $[CLS]$ token is also used in the following similarity and paraphrase tasks:

- MRPC (Microsoft Research Paraphrase Corpus, Dolan and Brockett (2005) [141]): its corpus consists of sentence pairs from news websites with annotations regarding their semantic equivalency

- QQP (Quora Question Pairs, Chen et al. (2018) [142]): its corpus is a set of question pairs from the Quora QA community with annotations regarding their semantic equivalency

- STS-B (Semantic Textual Similarity Benchmark, Cer et al. (2017) [143]): its corpus is a collection of sentence pairs from news headlines, video and image captions and NLI data annotated with similarity scores ranging from 1 to 5

The same setup is used by the following inference tasks:

- MNLI (Multi-Genre Natural Language Inference, Williams et al. (2018) [144]): its corpus contains a set of sentence pairs collected with crowd-sourcing and followed by textual entailment annotations, i.e. given a premise sentence and a hypothesis, the task is to predict whether the premise entails the hypothesis, contradicts it or neither

- QNLI (Question Natural Language Inference, a version of the Stanford QA Dataset, Rajpurkar et al. (2016) [145]): a QA dataset that is modified into QA sentence pairs annotated with whether the second sentence is the answer to the question or not

- RTE (Recognizing Textual Entailment, (Bentivogli et al. (2009) [146]): its corpus comes from annual textual entailment challenges. The task is similar to MNLI but with much less data

- WNLI (Winograd NLI Schema Challenge, Levesque et al. (2011) [147]): a entailment task with a corpus made of annotated sentence pairs. It is not used by Devlin et al. (2018) [11]

Devlin et al. (2018) [11] also used fine-tuned their models on the SQuAD v1.1 and v2.0 tasks, and on the SWAG dataset. SQuAD v1.1 (Stanford Question Answering Dataset, Rajpurkar et al. (2016) [145]) is a set of 100k crows-sourced sentence-paragraph pairs. The sentence is a question, that is answered by a span of words of the paragraph. The goal is to find the correct span in the paragraph. The method is presented in chapter 3.9.2 and shown in figures 3.6 and 3.7 is used. Instead of two FFNNs, a start vector $\mathbf{S}$ and an end vector $\mathbf{E}$ are learned and their dot product with the encoder outputs corresponding to the positions of the tokens of the paragraph $\mathbf{o}_i$, $\forall i \in \{1, \cdots, L_{par}\}$, is computed. Softmax layers assign two probability values to each token computing the probability of each one being the first token $p_{start,i} = \frac{exp(\mathbf{S} \cdot \mathbf{o}_i)}{\sum_{j=1}^{L_{par}} \mathbf{S} \cdot \mathbf{o}_j}$, $\forall i \in \{1, \cdots, L_{par}\}$, and the final token $p_{final,i} = \frac{exp(\mathbf{E} \cdot \mathbf{o}_i)}{\sum_{j=1}^{L_{par}} \mathbf{E} \cdot \mathbf{o}_j}$, $\forall i \in \{1, \cdots, L_{par}\}$ of the span respectively. The highest scoring pair of tokens, $\mathbf{S} \cdot \mathbf{o}_s + \mathbf{E} \cdot \mathbf{o}_e$, $s \leq e$, is chosen.

SQuAD v2.0 also contains sentence-paragraph pairs, where the paragraph does not contain the answer to the question asked. The $[CLS]$ token acts as an indicator as to whether the paragraph contains the answer.

SWAG (Situations With Adversarial Generations, Zellers et al. (2018) [148]) dataset is a collection of 113k sentence-pair completion examples. The task is to choose the most probable continuation for a sentence among four options. For each example they create four model inputs, each one consisting of the sentence and a probable next phrase. They introduce a task-specific vector C that is multiplied with each of the four resulting $[CLS]$ vectors. This leads to four numbers that are given as inputs to a softmax layer that performs the classification.

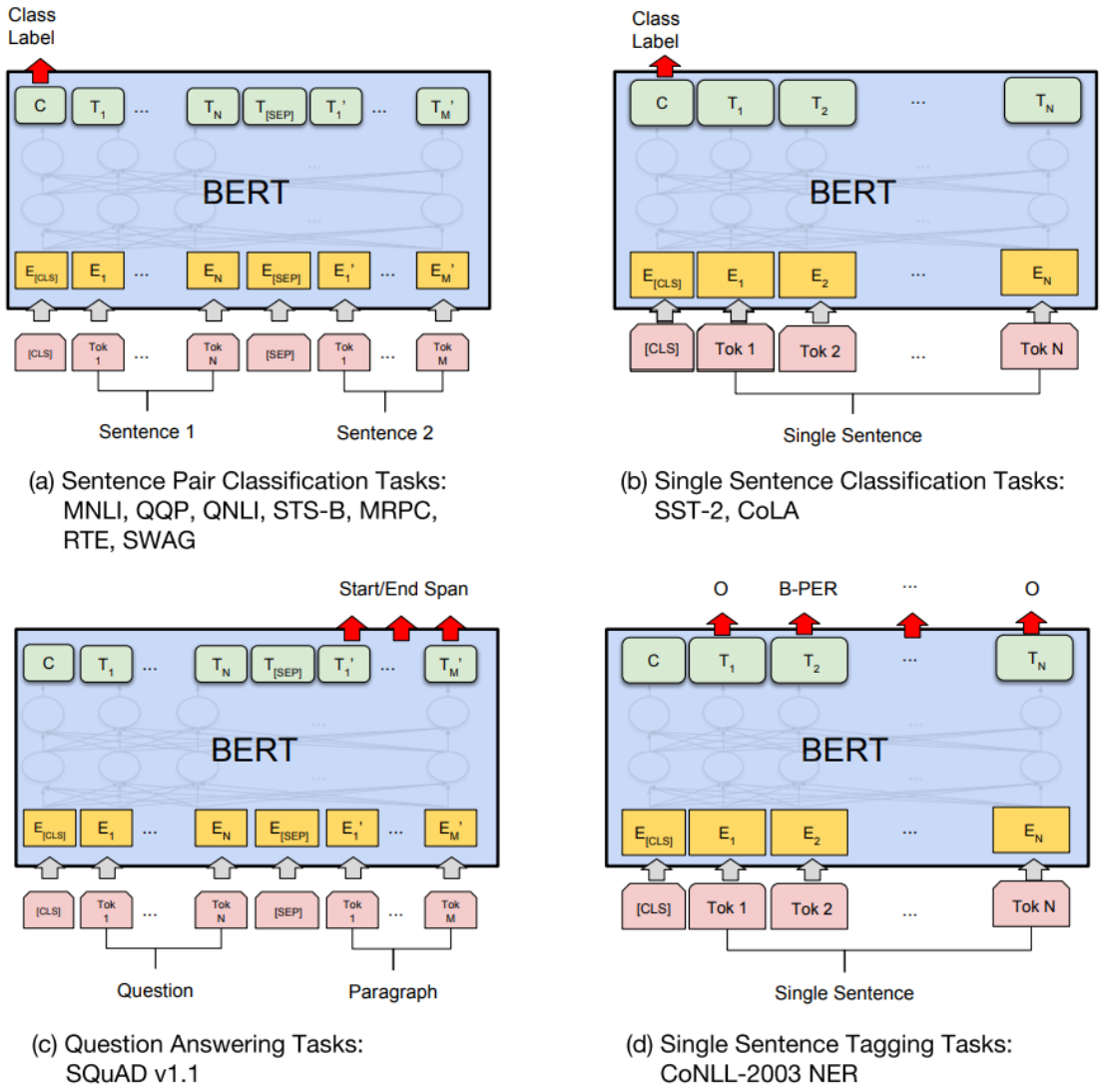


Figure 3.7. *[11]*

Results

Both the base and the large model outperformed the next best model in the GLUE benchmark by 4.5% and 7.0% respectively. Their large model also outperformed all other models in the SQuAD tasks by a clear margin and outperformed GPT in SWAG by 8.3%.

In their ablation study (table 3.2) they prove the importance of NSP for handling tasks with two input sentences as well as the importance of using a deep bidirectional net for creating contextualized representations.

Tasks	MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (Acc)
$\text{BERT}_{\text{BASE}}$	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

Table 3.2. *Ablation study performed to $BERT_{BASE}$. The study shines light on the importance of the NSP task and of bidirectionality.*

3.10 Overparameterization of the Heads of Transformer-based Models

Cordonnier et al. (2021) [13] find indications of overparameterization in the heads of the Transformer architecture. They suggest that examining the rank of the multiplications of key and query matrices of each head separately, $\mathbf{W}_i^Q(\mathbf{W}_i^K)^T \in \mathbb{R}^{D_h \times D_h}, i \in \{1, \cdots, H\}$, does not suffice to reveal the issue, which lies in the commonalities among the sub-spaces attended by every head. The is evident in the left side graph of the figure 3.8, where the red line ascends smoothly as the number of considered dimensions increases, indicating that the multiplication of the associated matrices is not low-rank in general.

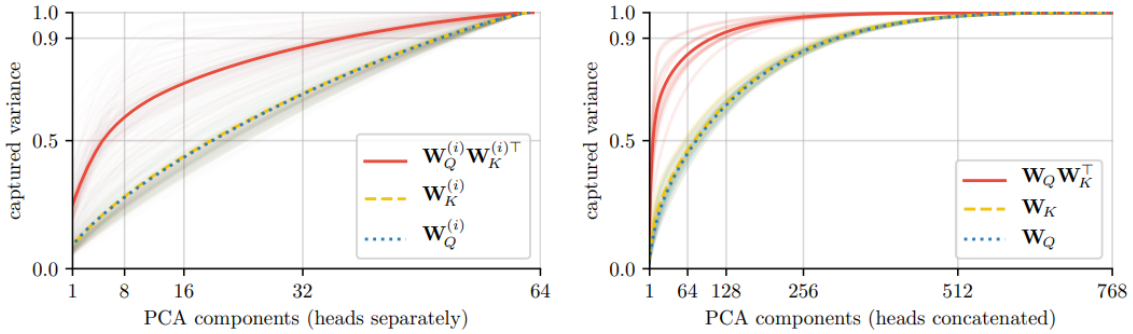


Figure 3.8. *The matrices are taken from a pre-trained BERT model with $H = 12$ and $D_{key} = 64$. PCA [12] is performed the cumulative variance w.r.t. the number of dimensions used is shown in the graphs. The left graph presents this metric separately for each head and the right one presents it for the matrices produced by the concatenation of the respective head-specific matrices [13]*

They thus concatenate head-specific matrices into two new ones, $\mathbf{W}^Q$ and $\mathbf{W}^K$, and perform PCA to $\mathbf{W}^Q(\mathbf{W}^K)^T \in \mathbb{R}^{D_{in} \times D_{in}}$, $(D_{in} = D_h \cdot H)$. This matrix is evidently low-ranked, which means that the flexibility provided with the separation of the attention process into heads is underutilized.

3.11 Scheduled DropHead

Zhou et al. (2020) [14] seek to mitigate the problem of attention head under-utilization. Based on the work of Voita et al. (2019) [16] (chapter 4.4.2) and Clark et al. (2019) [17] (chapter 4.4.3), they contend that the issue arises for two reasons; First some attention heads tend to be used much more than others, i.e. become **dominant**, a matter related to the problem of module collapse discussed in chapter 4.12.2. Second, attention heads **co-adapt**, i.e. avoid learning what other heads already know.

3.11.1 DropHead

In order to solve this problem they introduce an attention head regularization technique, called **drophead**. Drophead is based on the idea of the dropout regularization technique addressed in chapter 2.4.5, that involves randomly masking neural units during the training of the NN to prevent co-adaptation between neurons and promote the creation of multiple well-trained neural paths. Zhou et al. (2020) [14], instead of dropping neurons, drop entire attention heads. Specifically, they sample a random mask vector $\boldsymbol{\xi}_l \in \{0, 1\}^H$ and apply it to the outputs of the attention heads of the l-th layer, $l \in \{1, \cdots, L\}$, every time a new input sequence is used:

$$\boldsymbol{\xi}_l = \{\boldsymbol{\xi}_{li}\}_{i=1}^{H},\ \{\boldsymbol{\xi}_{li}\} \in \mathbb{R}^{D_{val}} \tag{3.25}$$

Each mask is applied to the corresponding head's output via element-wise multiplication. Similarly to the dropout method, they normalize the output of the attention heads by dividing the result with $\gamma = \text{sum}(\boldsymbol{\xi}_i)/H$ in order to ensure the matching of scales between training and inference times. This mechanism is shown in figure 3.9.

Since dominant heads are unavoidably masked for a large set of training samples, the rest of the heads are forced to learn how to make up for this fact. The same is true when it comes to the co-adaptation problem, as not head can rely on features learned by another one. This effectively reduces overfitting and boosts generalization performance.

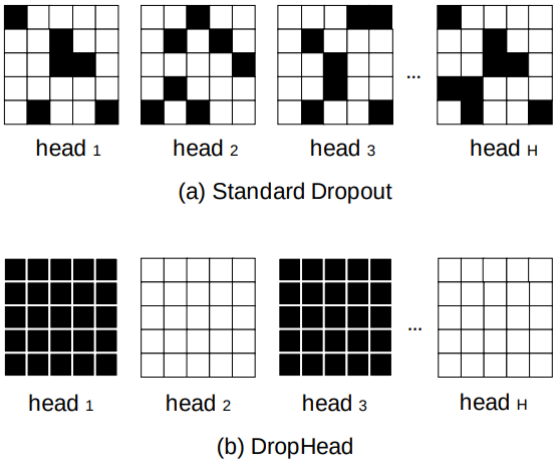


Figure 3.9. *Examples of* ***(a)*** *the application of a standard dropout technique that randomly drops a set of neurons and* ***(b)*** *the DropHead method that randomly drops entire heads [14]*

3.11.2 Scheduler

Is is common to use a dropout scheduler that linearly increases the dropout probability as training proceeds (curriculum schedule). Doing so effectively prevents co-adaptation, that has been found to become worse at the end of training, while ensuring that neural structures are sufficiently trained.

Zhou et al. (2020) [14] propose a V-shaped scheduler, which starts head dropping with a relatively high probability p_{start}, that linearly drops to 0 and then linearly increases up to p_{end}, as shown in figure 3.10. Specifically they use $p_{start} = p_{end} = 0.2$. They contend that starting of with a high drophead probability reduces the risk of few heads dominating the multi-head attention mechanism, which happens right from the begging of the training process as observed by Michel et al. (2019) [149].

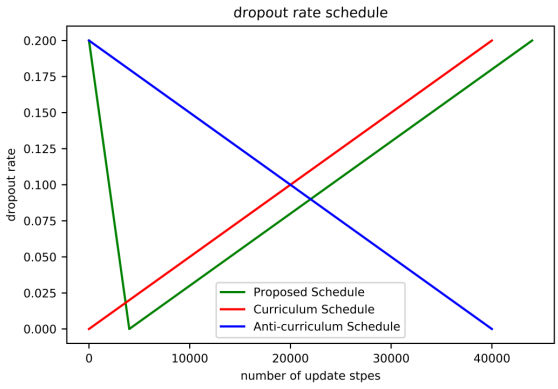


Figure 3.10. *Several dropout schedulers. The curriculum scheduler (red) linearly increases dropout probability as training proceeds while the anti-curriculum scheduler (blue) does the opposite. The proposed scheduler (green) linearly decreases the dropout probability and then linearly increases it [14]*

3.11.3 Experiments

They apply their method to the transformer model, which they train on a NMT task, and to BERT, which is trained on text classification tasks.

NMT with Transformer

They train several variants of the big transformer model, each one with a different dropout-based technique:

- the standard dropout method applied to the neurons of the attention heads
- the drophead method w/o the use of a scheduler
- the scheduled drophead method

They attribute the obvious improvement in performance, presented in table 3.3, to a success of their method in preventing the overfitting of the model's attention heads. They thus try increasing the number of attention heads, from 16 per layer to 32 per layer, while simultaneously decreasing the number of neurons per head to maintain a constant model size. This further boosts the results of their model , but not of the standard transformer implementation, proving their point.

Models	BLEU	PPL
Weighted Transformer	28.9	-
Tied Transformer	29.0	-
Layer-wise Coordination	29.1	-
Transformer-big	28.4	-
ours (reproduced)	28.7	4.32
+ Attention Dropout	28.7	4.29
+ DropHead	29.2	4.15
+ Scheduled Drophead	29.4	4.08
Transformer-big (more heads)	28.4	4.39
+ Attention Dropout	28.5	4.38
+ DropHead	29.3	4.12
+ Scheduled Drophead	**29.6**	**4.02**

Table 3.3. *Performance of various models on the NMT task WMT14 (en-de) [30]. They reproduce the original Transformer model and report results for it too. They compare to the results of the weighted transformer [31], the tied transformer [32] and the layer-wise coordination method [33] applied to the transformer model.*

Text Classification

They apply their method during the fine-tuning of the BERT model in several text classification tasks and evaluate its performance. Improvements are observed but are not significant. Since BERT is already pre-trained before drophead is applied, they contend that the attention heads have already been formed and thus the room for improvement is limited.

They therefore initialize new transformer-based models and train them on the text classification tasks, without LM-based pre-training, while applying their methods, in order to evaluate their contribution to the the performance of a model that is trained from scratch. They indeed report substantial improvements over the transformer-based baseline.

Effects of DropHead

To test whether scheduled drophead truly minimizes the dependence of the transformer model on dominant heads they try deactivating the head of each layer that leads to the maximum drop in performance, one head at a time, and then report the average performance over its layers. They observe that the effect of masking the dominant heads of a model trained with scheduled drophead is limited compared to one of the vanilla transformer model (table 3.3). This means that scheduled drophead effectively limits the dependence on individual heads and forces more heads to contribute to the output.

Models	Enc-End	Enc-Dec	Dec-Dec
Transformer	-0.47	-1.05	-0.32
+ DropHead	-0.31	-0.79	-0.27
+ Scheduled DropHead	-0.28	-0.70	-0.25
Transformer (more heads)	-0.41	-0.92	-0.29
+ DropHead	-0.25	-0.61	-0.24
+ Scheduled Drophead	**-0.21**	**-0.54**	**-0.20**

Table 3.3. *Average drop of the BLEU score after the most important head of a layer has been removed across layers.*

Chapter 4

Modeling System 2 with Neural Networks

4.1 Introduction

Modern neural networks are able to learn complicated patterns in a matter of days. They can chat like humans [150], perform certain image recognition tasks better than humans [65], and have started helping scientists in solving problems that have remained open for decades [151]. Nevertheless, they are not yet capable of performing tasks that the human brain excels at, like systematically generalizing, i.e. combining pieces of already acquired knowledge to solve tasks they have never seen before.

Part of the research effort on neural networks now focuses on enabling them to acquire such capabilities that only the human brain is known to possess. In chapter 2.4 the contribution of neuroscientific findings in the creation of NNs was highlighted. The study of synapses, neural activation and of plasticity were crucial in providing insight for the development of the notions of neural weights, activation functions and learning processes for NNs.

Another re-occurring theme in this thesis is the successful transformation of prior beliefs about the world into model architectures (chapter 2.5.2. The resulting techniques offer significant computational and expressive advantages over standard NN architectures. Such are the methods of weight sharing and sub-sampling and the mechanisms of convolution and attention that are used by modern NNs and were inspired by prior beliefs such as equivariance over space and permutations.

A research team, lead by the computer scientist Yoshua Bengio, is trying to use intuitions from neuroscience as prior beliefs to create a generation of neural networks that will better mimic the way the human brain works and will exhibit characteristics that modern NNs don't, like being able to systematically generalize.

Their research is heavily influenced by the works of:

- Daniel Kahneman & Amos Tversky: psychologists that studied cognitive biases and decision making
- Bernard Baars: neuroscientist, mostly known for the Global Workspace Theory regarding the way cognitive abilities and consciousness function
- Bernhard Schölkopf: computer scientist known for his work on causality and kernel methods

In order to present the vision of Yoshua Bengio and his team, relevant aspects of the works of the above researchers as well as closely related research areas will first be discussed.

4.2 Thinking In Different Speeds: System 1 And System 2

In his book, Thinking, Fast and Slow [50], Daniel Kahneman adopts a terminology first proposed by the psychologists Keith Stanovich and Richard West. He uses the terms **System 1** and **System**

2 as simplifications to talk about the two functioning modes of the human brain.

System 1, he contends, operates involuntarily, quickly and without the human's control. This is the system that performs trivial arithmetic operations like 2×2, recognizes the emotions of people by a simple look at their faces and creates an internal image of a spoken object. It is therefore responsible for creating impressions, and in fact, for maintaining a model of the world as it is viewed by the brain's owner. This model entails people, objects, ideas and relations between them that the person views as normal and usual and thus there is no need for mental effort in order to represent them.

Impressions system 1 generates may be the result of what Kahneman refers to as **heuristics**. For example, humans estimate the probability of an event by relying on the ease with which relevant events come to mind, e.g. knowing many divorced couples may lead to overestimating the probability of a marriage leading to divorce. Kahneman, among many psychologists, believes that the use of heuristics is a reason for **predictable biases**, i.e. systematic errors that are made by humans.

Not all impressions generated by System 1 result from intuitive heuristics. **Expertise** accomplished by repetitive practice is distilled as a System 1's response to a familiar situation. This is how a basketball player instantaneously reacts to a shot attempted by an opponent or a Scrabble master in a newly formed word.

Nevertheless all impressions created by System 1 initiate from the recognition of a situation that is somehow familiar. System 1 also detects situations that are novel to the human and are not explicitly represented in memory. Such are the cases of nontrivial arithmetic operations, like 57×36, the task of playing a new game or the task of writing a thesis. These cases violate the model maintained by System 1, and demand the human to willingly place effort to handle them. Such tasks activate System 2, which is much slower than System 1 but, in contrast to it, can devise and implement a plan consisting of a series of distinct steps.

System 2 is what humans identify themselves with, an existence conscious of itself, with its own feelings, beliefs and ideas that can manipulate thoughts to reason about problems and situations. System 2 requires conscious attention and effort to function, the degree of which depends on the difficulty of the task at hand. This is why it is usually in a low-effort mode and not full on. System 1 continuously provides System 2 with impressions that it ordinarily accepts, turning them into beliefs. But, when faced with a new situation, System 2 is called to carefully examine System 1's impressions and suggestions, decide what is valid and relevant and then synthesize a response to the novel conditions. In contrast to System 1, System 2 is capable of statistical thinking and is therefore frequently called upon to reconsider some of System 1's biased judgements.

System 2 can also guide System 1 into behaving in different ways than it normally does. It can program its attention to focus on recognizing new sets of patterns, like a specific word in a test, and the associative memory to retrieve data that are related to a specific theme, like landlocked countries. Moreover, re-occurring tasks that initially call for System 2's attention gradually become easier and require less attention and effort as a significant portion of steps begin being handled automatically by System 1.

4.3 Modularity In The Human Brain

4.3.1 Introduction

The neurons of the human brain have been found to compose what is known as **small-world networks**. The vertices of a small-world network are not all connected to one another, but the length of the shortest path between the neurons is relatively small ($L \sim \log N$, N the number of

vertices in the graph) [71]. The boundaries between different areas of the human brain are mostly not well defined and cognitive functions cannot be localized to specific brain areas. Yet, research indicated that parts of the brain, called **brain nuclei**, do exhibit signs of specialization in terms of their functionality. This specialization of brain nuclei, as clusters of neurons with distinct sets of functions are called, is a possibly important source of inspiration for the creation of the next generation of neural networks. It is thus useful to become familiar with the basic areas of the human brain and with the way they are segmented based each one's different functionality. During this discussion the most intriguing brain nuclei, such as the basal ganglia, will also be examined and key aspects of of the matters of consciousness and attention will be highlighted.

The neurons of each brain area have been found to work together to perform the necessary functions. Neuroscientists have been able to link each area to a specific set of functions by using three methods, i.e. by associating damages in a brain area with the impaired function, by following the neural paths, and by observing brain activity with brain scanning techniques. They have thus split the brain into three main areas, the forebrain, the midbrain and the hindbrain. Each one is further divided into specialized nuclei. The **midbrain** is a relatively small brain area involved in the sleep-wake cycle, thermoregulation and visual reflexes [71].

4.3.2 Forebrain

The **forebrain** accounts for 90% of the brain's mass and is involved in higher cognitive functions and the perception of sensory inputs. It includes the **cerebrum**, which in turn contains the **cerebral cortex** that is related to functions such as consciousness, language and memory. The cerebral cortex is the layer that surrounds the brain. The segmentation does not end here since the cerebral cortex consists of 4 lobes, each one with its own specialization:

- **temporal lobe**, specialized in language and emotion
- **occipital lobe**, involved in vision
- **parietal lobe**, responsible for processing sensory signals related to touch and for body posture awareness
- **frontal lobe**, associated to short-term memory

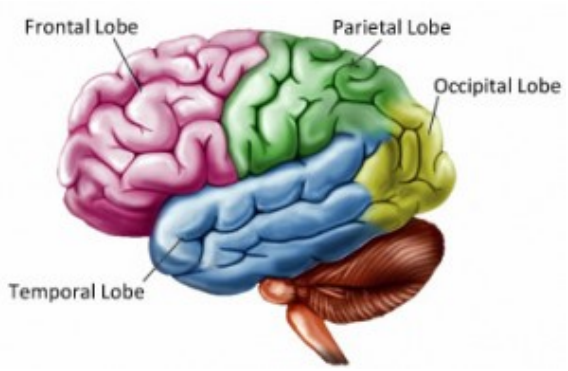


Figure 4.1. *A figure indicating the 4 lobes of the forebrain. Figure from* `https://www.nbia.ca/brain-structure-function/`

Each of the lobes contains several areas with different functionalities. The inferior temporal gyrus, for example, which is part of the temporal lobe, is associated to face face recognition, and the orbitofrontal cortex, that belongs to the frontal lobe, is involved in emotion representation and reward generation in decision making [71]

Language

One can also spot a set of areas that play important roles in language processing, understanding and generation:

- **Auditory cortex**: part of the temporal lobe located at the side of the brain that processes auditory information

- **Angular gyrus**: part of the inferior parietal lobe that is responsible for linking words to images, thoughts and feelings. It is thus an area where information from multiple senses is gathered and processed jointly to create a thorough representation of even complex notions and concepts
- **Wernicke's area**: part of the temporal lobe that assists the comprehension of language and the process of choosing words when speaking
- **Broca's area**: part of the frontal lobe that has been found to be active when learning a new language and is also involved in speech generation. In fact, different parts of it are used when the person speaks different languages

These areas are shown in figure 4.2.

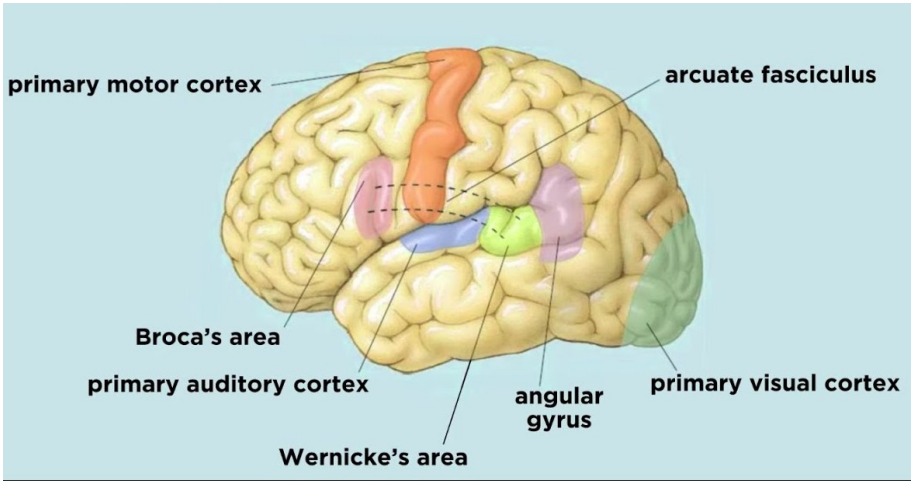


Figure 4.2. *A figure indicating the areas of the brain connected to language understanding, processing and generation. Figure from* `https://www.youtube.com/watch?v=zj0yud4wv74`

Limbic System

At the forebrain's base, located between the cerebral cortex and the midbrain, is the **thalamus**. The thalamus, along with the structures that surround it, assists the transmission of signals between the forebrain and the brainstem, while also providing a connection to the rest of the body. Such are signals from every sensory system, except smell. It is also involved in sleep, alertness and consciousness.

The **hypothalamus** is located below the thalamus and connects the brain to the endocrine system. It does that by synthesizing and releasing *neurohormones*. It controls aspects of growth and homeostasis and plays key roles in actions such as drinking and eating.

The **pituitary gland** weights only 0.5 g and is found beneath the hypothalamus. Under the latter's control, the pituitary gland produces hormones related to growth, urination, reproduction etc [71]

These areas are shown in figure 4.3.

Basal Ganglia

The **basal ganglia** is a set of nuclei located in the cerebrum that have a very interesting role. Due to the brain's complexity it is very usual that different areas issue commands that come in direct conflict with each other. The role of the basal ganglia is to receive the respective signals

and decide which commands will be followed. In order to achieve this goal they use multiple paths with feedback loops that allow some signals to pass while inhibiting others. Three basic paths can be distinguished whose routes depend on the origins of signals each one is meant to meant to block or allow passage to. For example, the motor loop is responsible for selecting muscle actions and is thus connected to the centers controlling movement. The other two are the prefrontal loop and the limbic loop [71]. A few notable components of the basal ganglia are:

- the **caudate nucleus** that is involved in motor processes, procedular learning, associative learning and conscious inhibition of actions
- the **putamen** that is involved in complicated motor behaviours, motor planning, learning and execution
- the **substantia ganglia** associated with eye movement, motor planning and reward seeking

4.3.3 Hindbrain

The **hindbrain** is the oldest part of the brain as its genes were formed around 560 million years ago. It consists of the cerebellum and the brainstem [71].

Cerebellum

The **cerebellum** drives signals carry commands related to any type of body movement through its multilayered neural paths. The role of these paths is to translate high-level commands into delicate sequences of muscle contractions. The pattern of contractions for each movement is stored in the way these neural paths are formed. This multilayered setup played a significant role in inspiring artificial multilayered neural networks.

The cerebellum is indicated in figure 4.3.

Brainstem

The **brainstem**, shown in figure 4.3, is directly connected to the spinal cord. It thus serves as a hallway for neural signals. It is also home to various nuclei, such as the reticular formation associated to alertness and consciousness and the medula that is in charge a variety of autonomous body functions like modulating blood pressure.

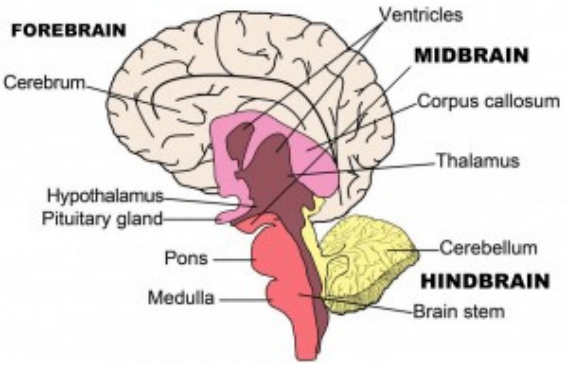


Figure 4.3. *A figure indicating several important areas of the brain. Figure from* `https://www.youtube.com/watch?v=zj0yud4wv74`

4.3.4 Consciousness

Consciousness is the awareness of the external world's aspects that can sensed through the sensory organs and the internal world's happenings like thought and emotions. Modern neuroscience is able to track brain processes that are known to be related to consciousness and some of them occur in the aforementioned brain areas. But, the source of this phenomenon has not yet been determined. That is, researchers don't know if consciousness comes as a result of this activity or the two are simple connected somehow [71].

4.3.5 Attention

Attention is the conscious decision to focus on a particular stimuli or internal event. Deciding to pay attention to a stimuli activates the brain regions that are associated to the related sense. For example, the parietal lobe that handles spatial data may activate to guide the frontal lobe into instructing the eyes to look at a particular spot of interest. It is important that each person has a limited "attention budget" and the brain cannot be ordered to pay attention to more than a few things simultaneously [71].

This type of system configuration that is involves the specialization of different subsystems in distinct sets of functions is not only not only found in the human brain, but throughout nature. This property is so common that it has a name, i.e. **modularity**, while systems that exhibit this type of behaviour are called **modular** and the specialized subsystems are called **modules**.

4.4 Modularity In Neural Networks

Even though a mechanism that explicitly promotes modularity in NNs has not yet been introduced in this thesis, some of the models that have been presented have been found to acquire modular properties solely via the training process. The most prominent cases are the ones of CNNs and of Transformers.

4.4.1 Modularity In CNNs Trained On Scene Classification

Zhou et al. (2015) [14] found that the kernels of a CNN, that is trained on a scene classification problem, become specialized in detecting objects that are related to the scene categories the model has witnessed. For example this occurs in the case of a bedroom scene that leads to the specialization of some kernels at detecting bedroom-related objects like beds and lamps.

Zhou et al. (2015) [14] used a CNN trained from scratch on 2.4 million images from 205 scene categories. They then selected images that induced the biggest activations for each several units across the layers of the CNN. For each image they generate 5000 versions, they obscure in each one a random part of the image and then record the change in the induced activation of the respective neural unit. This is done in order to locate the object in the image that is most strongly connected to the activation of the unit. The resulting objects were shown to human annotators who were asked to categorize each unit based on the objects that were found to activate it the most. The units were also categorized based on the semantic level of the concept that activates them, ranging from low-level concepts such as simple elements and colors to high-level concepts like objects and scenes. The annotators also were told to record for each unit the number of images that contain concepts that seem to activate it but deviate from the unit's concept category. The percent of images for each unit that contain objects that do belong to the unit's semantic label and indeed activate it is named as the unit's *accuracy*.

Around 60% of the units at each layer has over 75% accuracy, based on the above metric, suggesting that kernels are indeed specialized in identifying specific concepts. In addition to that, units at early layers were found to focus on low-level concepts while units at later layers identify high-level concepts, as shown in figure 4.4. This is in accordance with Funahashi (1989) [87] and Chester (1990) [88] 2.4.5) that found that neurons of the first layer of a 2-layer NN extract local features while the neurons of the second layer use the local features to extract global ones.

Moreover, certain re-occurring objects were found to activate more than one unit. In fact, there are cases where units are further specialized in the various forms different of a specific object, e.g. 6 units were found to detect lamps and each unit to be responsible for detecting a different lamp

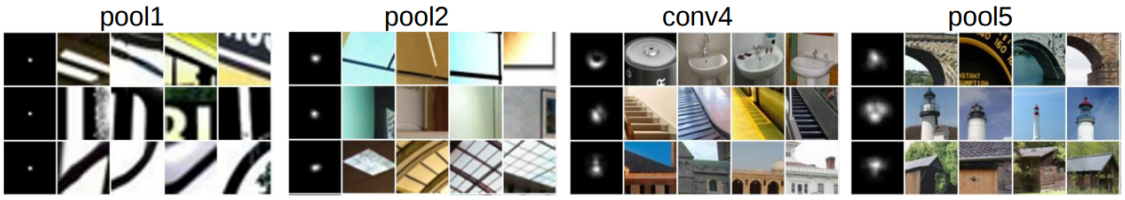


Figure 4.4. *The receptive fields of 3 units of the layers pool1, pool2, conv4 and pool5 along with the images areas inside the receptive fields that activate these units the most [15]*

type.

4.4.2 Modularity In Transformers Trained On NMT

Voita et al. (2019) [16] studied the specialization of the attention heads in a Transformer model train on NMT data. They chose English as the source language which helped them to analyze the behavior of the encoder's self-attention heads. They chose Russian, German and French as target languages.

They first used **layer-wise relevance propagation (LRP)** to locate the most important heads. LRP estimates the importance of each unit starting from the output units and propagating backwards. It uses the weight ratio between units of consecutive layers as an importance propagator, while considering the total relevance to be constant across layers.

Voita et al. (2019) [16] attempted locating the most important heads by averaging the importance of their constituting neurons. They then manually tried to find where the most important heads pay attention to. To do that they observed where the maximum attention weight of each head is usually assigned to.

They found some heads that, at 90% of the time pay most attention to their nearby positions and especially to tokens right before and right after their respective position. These heads were named **positional** heads and were ranked as the most important ones by the LRP metric.

Other heads were found to focus on specific syntactic relations. More specifically, Voita et al. (2019) [16] looked for the following relations: nominal subject, adjectival modifier, direct object and adverbial modifier. They use the CoreNLP model (Manning et al. (2014) [152]) for syntactically parsing a set of sentence pairs that the Transformer model had not previously seen. Then, they calculated the frequency with which each head assigns its maximum attention following one of the aforementioned dependencies in either direction. If they found that the assignments of a head consistently follow one of these relations then the considered to be a **syntactic** one. Indeed, some of them were able to accurately predict syntactic relations with high accuracy indicating that they has acquired a specialized syntactic role.

Finally, a head in the first layer of all models was found to pay attention to the **rarest token** in 66% of the sentences and to one of the two rarest in 83% of the sentences indicating another specialization.

By using a loss that balanced attention head pruning and model performance they observed that heads with specialized roles were the last ones to be pruned. In fact, while decreasing the number of active heads they discovered that the roles of the specialized heads that were pruned migrated ones that were still active (figure 4.6). This verifies the importance of specialized heads, as losing them leads to severe degradation of the model's accuracy (figure 4.7). Finally, the fact that the model initially retains its good performance despite the loss of many heads indicates that the rest of the heads that did not specialize were not as important as the specialized ones.

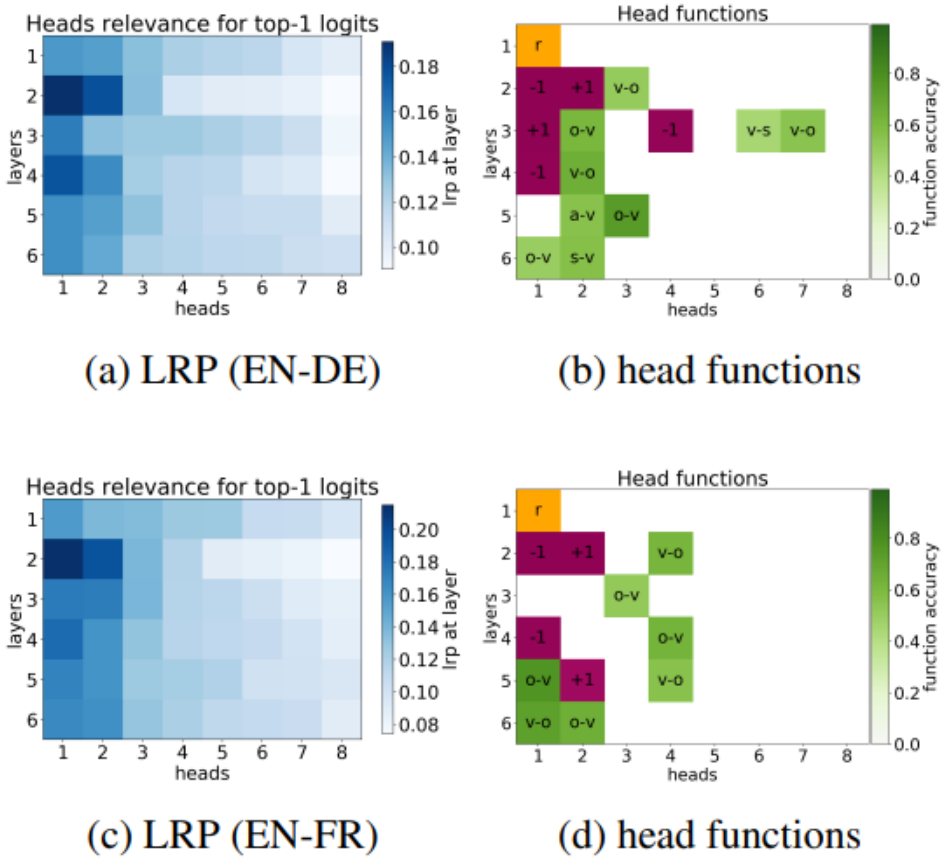


Figure 4.5. *The head relevance of the self-attention heads of the encoders of two models across all 6 layers and their corresponding specializations [16]*

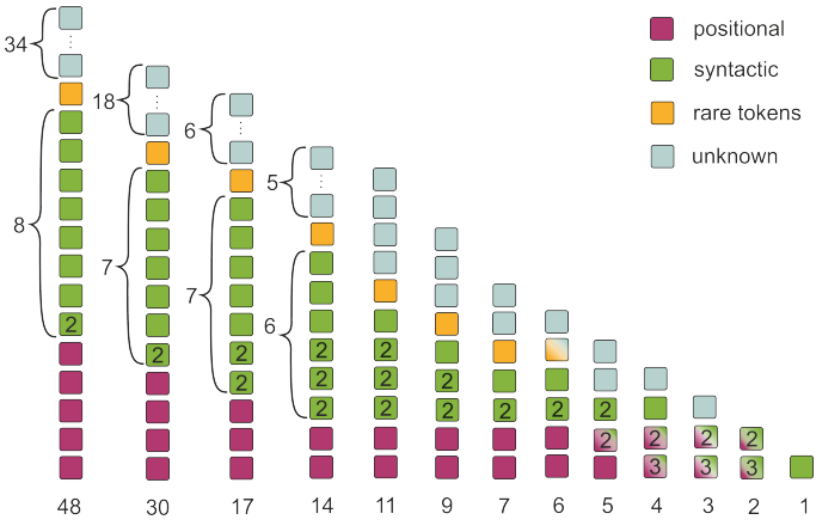


Figure 4.6. *The head relevance of the self-attention heads of the encoders of two models across all 6 layers and their corresponding specializations [16]*

4.4.3 Modularity In BERT Trained on NLM

Clark et al. (2019) [17] investigate the 144 attention heads maps of the $\text{BERT}_{\text{BASE}}$ model to discover patterns of positional or syntactic context. Recall that multi-sentence inputs to the

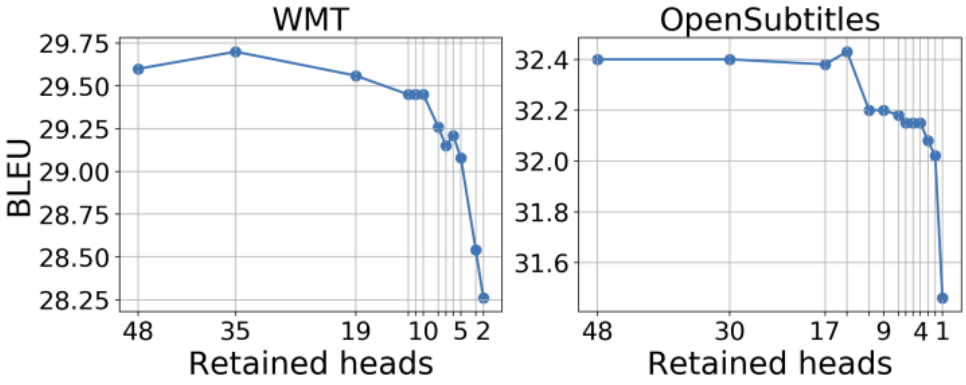
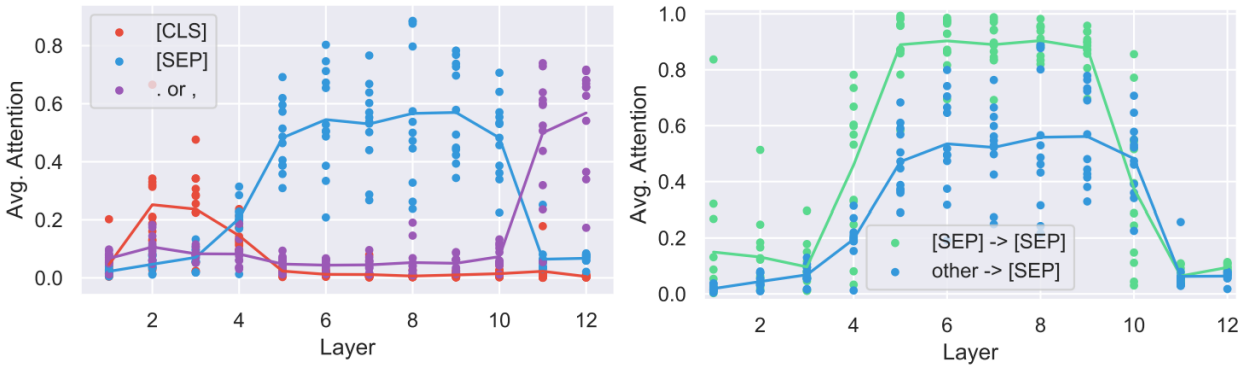


Figure 4.7. *The head relevance of the self-attention heads of the encoders of two models across all 6 layers and their corresponding specializations [16]*

BERT model are of the form: $[CLS]$ < paragraph 1 > $[SEP]$ < paragraph 2 > $[SEP]$. Clark et al. (2019) [17] extract attention maps over 1000 random segments from the 12 heads of each of the 12 BERT layers and then use them as data for their experiments. They refer to a specific head using the notation < layer >< head_number >.

They find that most heads don't pay much attention to the current token, but specialize in the previous or the next token, a behaviour also observed by Voita et al. (2019) (chapter 4.4.2). who refer to these heads as positional. In fact, they discover 4 attention heads in layers 2, 4, 7 and 8 respectively that on average place over 50% of their attention weight on the previous token and 5 attention heads in layers 1, 2, 2, 3 and 6 respectively that do so on the following token.

They also find that many heads, including more than half of the heads in layers 6-10, pay attention mostly to the $[SEP]$ tokens (figure 4.8), i.e. devote over 90% of their attention weight to themselves and the $[SEP]$ token. They hypothesize that heads with specific functions pay attention to $[SEP]$ in cases where their function is not applicable, e.g. a head that, under a direct object, pays attention to its verb, points to the $[SEP]$ token when it is found under a non-noun token.



(a) **(b)**

Figure 4.8. *Each points represents the average attention an attention head of the respective layer pays to* ***(a)*** *the corresponding token marked with red for the $[CLS]$ token, blue for the $[SEP]$ token, purple for commas and periods* ***(b)*** *the $[SEP]$ token marked with green if the head is found under a $[SEP]$ token and with blue otherwise [17]*

They use gradient-based measures of feature importance [153] to confirm this hypothesis. Indeed, starting from the fifth layer, they observe that, as the attention paid to $[SEP]$ increases,

the absolute value of the gradients that originate from it decreases, as shown in figure 4.9. This means that changing the outputs of the corresponding heads has small effects on the network's output, indicating that their minimal importance to the rest of the net when they are used that way. Importantly, it is also evident that when such a head is not applicable its parameters do not change much as a result of training.

In addition to that, they measure the entropy of head's attention distribution and show that attention heads in lower layers tend to simultaneously pay attention to a relatively large set of positions, i.e. devote at most 10 % of their attention mass to any single word. They therefore seem to gather information from multiple sources to create contextual representations, whereas heads in higher layers focus on few words each.

Moreover they study attention heads that are not deemed positional as to whether they perform syntactic roles. They discover that some attention heads do predict with considerably high accuracy scores specific syntactic relations while others do not systematically do so.

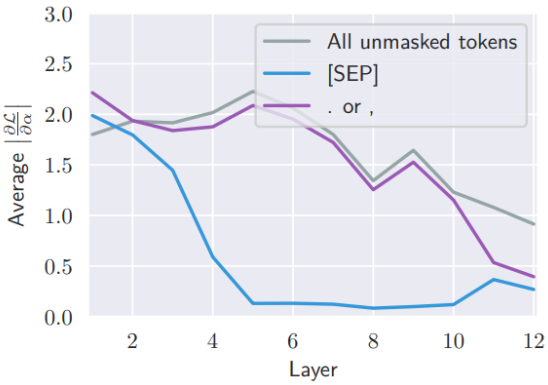
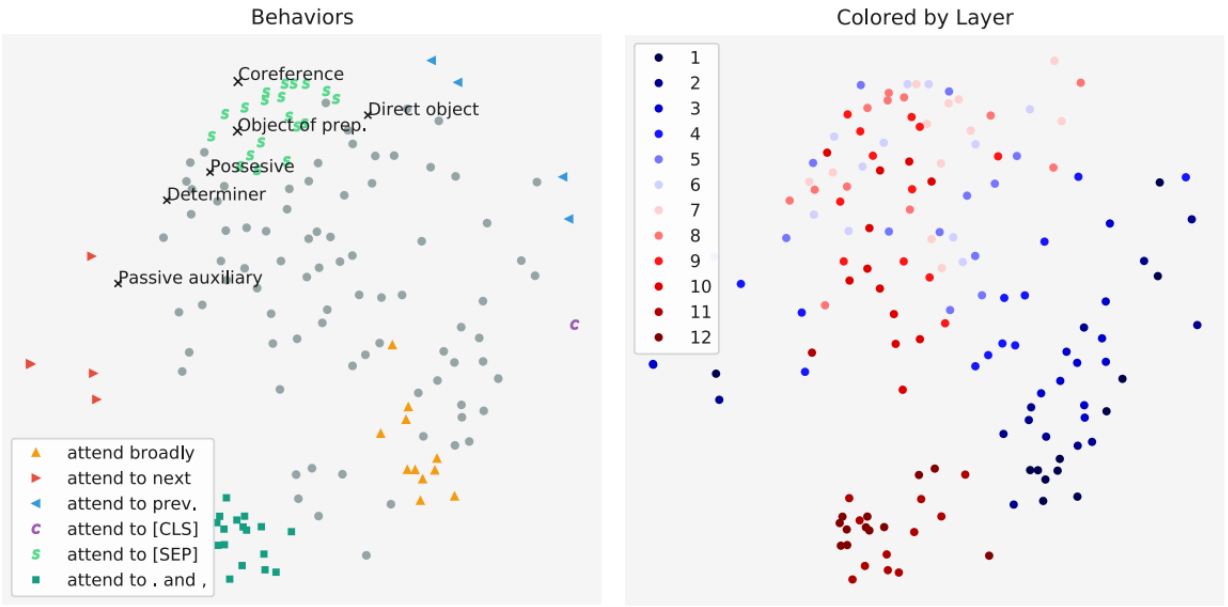


Figure 4.9. *Gradient-based estimation of feature importance for attention heads focusing on the [SEP] token, periods or commas and other tokens [17].*

Finally, they compute the Jensen-Shannon Divergence between attention distributions of each pair of heads to test whether one can group attention heads in terms of their functionality. They apply multi-dimensional scaling [18] to embed the heads in two dimensions based on their computed distances and present the results in figure 4.10.



(a) **(b)**

Figure 4.10. *Each point represents an attention head. The distance between each pair of attention heads is computed with the Jensen-Shannon Divergence between their attention distributions and multi-dimensional scaling [18] is used to embed them to a two-dimensional space. In* ***(a)*** *the heads are coloured based on their functionality and in* ***(b)*** *based on the layer in which they are found [17].*

Surprisingly, heads of the same layer seem to perform similar functions as captured by their attention distributions.

4.5 Global Workspace Theory

Bernard Baars proposed the **Global Workspace Theory (GWT)** [44, 45] in an attempt to explain how the distributed specialized brain nuclei synchronize and cooperate to support the brain's cognitive functions. He contends that there must exist a memory channel that can be accessed by all specialized processors or agents, as he calls the brain's distinct neural areas, to exchange information with each other and update on the current status. GWT suggests that it is consciousness that broadcasts data known by a single specialized agent to the rest of the brain areas. According to GWT, only conscious perception has access to the working memory and unconscious processes are confined near the respective specialized processors.

Baars [46] explained that, even though multiple events may cause an agent to send information to the channel, it is a selective attention system that decides whether to draw conscious attention to data reaching the channel or ignore them. This system, he states, is controlled by the frontal executive cortex and by areas that can automatically interrupt conscious processes, like pain and emotional centers. It acts as a bottleneck enabling only some important data to be broadcasted. Moreover, he highlighted that, after consciousness is informed of an event, it is conscious feedback that is necessary to control motor functions and some neural areas. Finally, he described the existence of self-executive interpreters, located in the frontal cortex and accessed by consciousness, that maintain high-level world information generating a feeling of consistency even when external situations change.

Baars et al. (2013) [47] and Baars and Geld (2019) [48] updated GWT to account for recent neuroscientific findings to create **Global Workspace Dynamics (GWD)**. GWD takes into consideration theories that view consciousness as the result of cortico-thalamic (C-T) activity. Baars et al. (2021) [154] defend the theory by explaining that GWT justifies this activity, that is believed to correspond to conscious experience.

4.6 Causality

The field of causality is a source of inspiration of the type of neural networks that will be discussed in this thesis. Before presenting the intuition from causality that is most relevant here, a few notes on causal models and interventions must first be provided.

4.6.1 Structural Causal Models

First the definition of the structural causal model with two variables will be given as it is sufficient for the following discussion.

Ορισμός 4.1. *A **structural causal model (SCM)** $\mathfrak{C}$ with two variables C and E and a graph with a edge emanating from C and ending in E, symbolized as $C \to E$, is given by the two assignments:*

$$C := N_C \tag{4.1a}$$

$$E := f_E(C, N_E) \tag{4.1b}$$

*where the random noise terms N_C and N_E are independent of each other. C is called a **cause** and E is called an effect. The graph $C \to E$ is called a **causal graph** and $\mathfrak{C}$ entails a joint distribution*

$p_{C,E}$ *over* C *and* E.

Ορισμός 4.2. *A* ***structural causal model (SCM)*** $\mathfrak{C}$ *with an arbitrary number of variables* D *is defined by a set of* D *structural assignments:*

$$X_i := f_i(PA_i, N_i),\ i = 1, \cdots, D \tag{4.2}$$

where N_i *is a random noise variable that is independent of all other noise variables* N_j, $j \in \{1, \cdots, D\} \backslash i$ *and* $PA_i \subseteq \{X_1, \cdots, X_D\} \backslash \{X_i\}$ *are the parents of* X_i *[19].*

4.6.2 Statistical vs Causal Learning

The goals of the models that were presented in chapters 2 and 3 are statistical learning goals, i.e. given observational data the models are trained to discover statistical quantities of the underlying mathematical model that generated the data. This problem is ill-posed Vapnik (1998) [155] because of the lack of information regarding the points (x, y) not contained in the dataset. This is why the Occam's Razor assumption (chapter 2.4.5) was introduced and as a result techniques that control the bias-variance trade-off, e.g. regularization, etc. A statistical model, as shown in chapter 4.6.1, is included in a causal model. **Causal learning** thus inherits the ill-posidness of statistical learning, but even complete knowledge of the underlying statistical model does not uniquely determine a SCM. It has been proven that observational data are insufficient to determine the causal graph as is proven by theorem 4.2.

Θεώρημα 4.2. *Every joint distribution* $p_{X,Y}$, *where* X *and* Y *are real-valued random variables admits SCMs in both directions.*

Proof. It always admits the SCM defined by the graph $X \to Y$ and the assignment

$$Y = f_Y(X, N_Y), \text{ where } X \text{ is independent of } N_Y \tag{4.3}$$

where f_Y is a measurable function (Peters (2012) [156]). Define the conditional cumulative distribution function:

$$F_{Y|x}(y) := P(Y \leq y | X = x) \tag{4.4}$$

Then define $f_Y(x, n_Y) := F^{-1}_{Y|x}(n_Y)$, where $F^{-1}_{Y|x}(n_Y) := inf\{x \in \mathbb{R} : F_{Y|x} \geq n_Y\}$. Then let N_Y be uniformly distributed on [0, 1] and independent of X. Since this is true independently of the structure of the model, the proposition has been proven. □

In order to learn a causal model someone must provide observation data generated by the model before and after a set of known changes in its structure. This matter will not further discussed here, but the interested reader is referred to Elements of Causal Inference, Peters et al. (2018) [19].

The power of causal reasoning makes up for the difficulty of causal learning, as the first one enables the analysis of the effect of interventions that will be discussed in chapter 4.6.3. As statistical learning is included is included in causal learning so is probabilistic reasoning in causal reasoning. This is elegantly shown in figure 4.11.

4.6.3 Interventions

An **intervention** to a variable means changing the corresponding assignment. For example, an intervention on variable E in equation 4.11 could be to set E to a constant value. This is symbolized as $do(E := e)$, where $e \in \mathbb{R}$ is that value. This also causes the entailed distribution

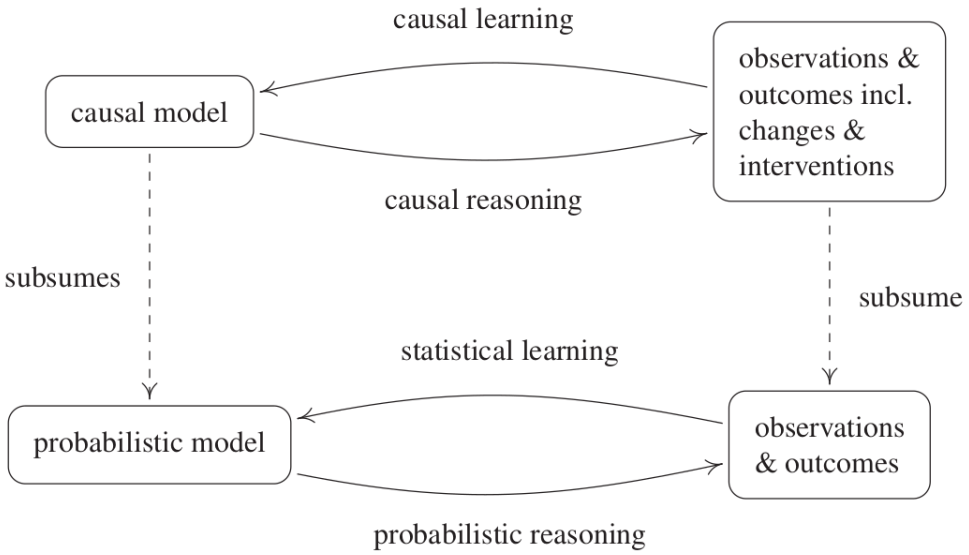


Figure 4.11. *Relations between the four processes and the resulting outputs [19]*

to change to $p_{\mathfrak{C}}^{do(E:=c)}$. This type of intervention is called a **hard intervention**. An intervention of the type $do(E := g_E(C) + \tilde{N_E}$, where g_E is a real-valued function and $\tilde{N_E}$ is a random noise variable, is called a **soft intervention**.

An example of a SCM is given by the following assignments:

$$C := N_C,\ N_C \sim \mathcal{N}(0,1) \tag{4.5a}$$

$$E := 3C + N_E,\ N_E \sim \mathcal{N}(0,1) \tag{4.5b}$$

that corresponds to the causal graph $C \to E$. Then $p_E^{\mathfrak{C}} = \mathcal{N}(0, 10)$. If a hard intervention is perform on C making equal to 2 ($do(C := 2)$), then $p_E^{\mathfrak{C};do(C:=2)} = \mathcal{N}(6, 1)$, which is also equal to $p_{E|C=2}^{\mathfrak{C}}$.

The important thing regarding interventions on causal models is that an intervention on the effect E does not change the distribution of the cause C. For example, $p_C^{\mathfrak{C};do(E:=0)} = \mathcal{N}(0,1) = p_C^{\mathfrak{C};do(E:=10)}$. But this is not equal to $p_{C|E=2}^{\mathfrak{C}}$, because in this case no one has intervened on E and therefore the original causal model is used. In $\mathfrak{C}$, knowledge about the possible values of the effect, E, indeed provides information about the value of its cause, C. An intervention can thus break causal relationships and it can also create new ones.

Interventions are similarly defined in the case of an SCM with multiple variables. Consider the case of a SCM $\mathfrak{C} = (S, P_N)$, where S is the set of assignments and P_N is the joint distribution of the noise variables. An intervention on $\mathfrak{C}$ is a change to one or more of the structural assignments. Intervening on a variable X_k can be done by using:

$$X_k := \tilde{f}(\tilde{PA_k}, \tilde{N_k}) \tag{4.6}$$

where one or more parameters have been changed. The new distribution is the symbolized as:

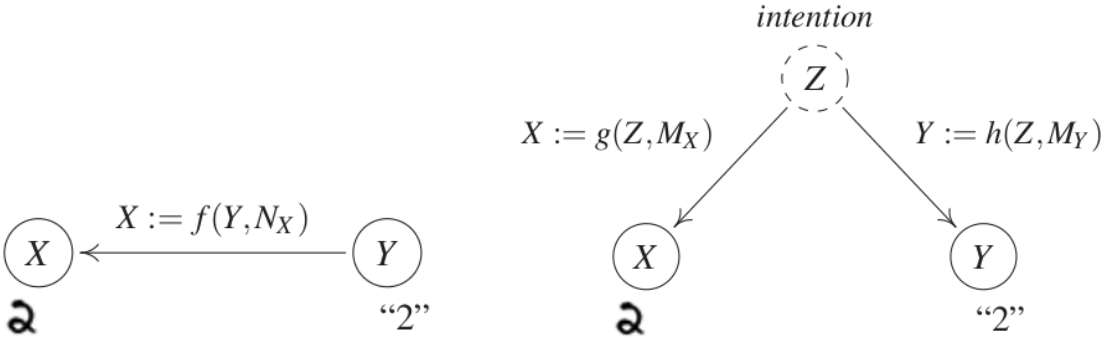
$$p_{\mathbf{X}}^{\tilde{\mathfrak{C}}} = p_{\mathbf{X}}^{\mathfrak{C};do(X_k:=\tilde{f}(\tilde{PA_k},\tilde{N_k}))} \tag{4.7}$$

4.6.4 An Example of A Causal Model

The following is an example given by Peters et al. (2018) [19]. Consider model (i) that generates pairs of images of digits with their corresponding labels as such: a human is given the number Y and is asked to write it down, creating X. If an intervention is performed on Y the corresponding label changes as the human sees a different number.

Now consider model (ii) in which the human is asked to think of a number and write it down. Now intervening on Y does not affect the drawing X.

Notice that the observational distributions are the same for both models. Yet, as it was shown, the same is not true about their intervention distributions. This was the point of chapter 4.6.2. This difference is depicted with the corresponding causal graphs in figures 4.12a and 4.12b [19].



(a) *Causal graph of model* (i) **(b)** *Causal graph of model* (ii)

Figure 4.12. *The causal graphs of the two causal models in the case of a sample with a label equal to 2. In model* (i) *the function* f *symbolizes the process of seeing the label* Y *and creating the corresponding image* X*. In model* (ii) *the random variable* Z *symbolizes the intention of the human to write down a number which then translates into a label through the function* h *and into a image of digit through the function* g *[19].*

4.6.5 The Principle of Independent Mechanisms

Consider now the case of the joint distribution function $p(l,t)$ of the latitude L and the corresponding temperature T, that has been calculated using samples from different latitudes taken across a constant longitude. There are two possible factorizations:

$$p(l,t) = p(l|t) \cdot p(t) \tag{4.8a}$$
$$= p(t|l) \cdot p(l) \tag{4.8b}$$

The first equation corresponds to the causal graph $T \to L$ and the second to $L \to T$. To decide which is the correct causal model, based on the above, one could intervene on a variable and check if that leads to a change in the distribution of the other. Obviously, changing the distribution of the latitude variable leads to a change in the distribution of the temperature one. However the opposite is not true; manually changing the temperature of a place does not change its geographic location.

Carefully examining the ability to locally intervene on causal model's variables without inducing a changing in the distributions of the other variables introduces another aspect of causal models. The mechanism that produces the distribution of latitudes $p(l)$ is independent of the mechanism that conditions on latitude values to produce temperature distributions $p(t|l)$ and thus intervening

on one does not affect the other. The individual mechanisms of a causal model are therefore autonomous, modular, or invariant. Daniušis et al. (2010) [157] refer to this principle, in the case of causal systems with two variables, as an **independence of cause and mechanism (ICM)**. Note that this principle does not apply to the mechanisms arising from the anti-causal factorization, $p(l|t) \cdot p(t)$.

At an information-theoretic level, the above can be translated into an independence of the information contained in the module that samples the cause variable from the information that characterizes the mechanism that uses it the cause to produce the effect variable. Knowledge about one of these modules cannot be used to infer knowledge about the other. Even though this is not a probabilistic argument, under certain assumptions it can also be assumed that the conditional densities that generate different causal variables are independent of each other.

Note that locally intervening on a mechanism $p(x)$ may affect the output distribution of the mechanism $p(y|x)$, where X and Y are causal variables connected via the causal graph $X \to Y$. But the mechanism $p(y|x)$ itself will not change.

4.6.6 Covariate Shift

New tasks that humans may be faced with are almost never entirely new to them. Instead, they are able to use previously acquired knowledge that they understand is still relevant in the new task. Only a small part of the knowledge required to perform the new tasks is usually learned especially for their needs. This radically increases the efficiency both in terms of required time and memory.

The principle of independence of cause and mechanism (ICM) could thus form the theoretical foundation for techniques that would allow neural models to acquire similar capabilities. One can assume that, during the formation of a new task from old ones, only few of the mechanisms that make up the underlying system are affected, while the rest remain unchanged. In the example of a causal system with two variables, if p_{cause} changes to p'_{cause} this does not means that $p_{\text{effect|cause}}$ is also different. Yet, even if it is, information about the way in which p_{cause} has changed is not relevant to the case of $p_{\text{effect|cause}}$. Therefore, using $p_{\text{effect|cause}}$ as an initialization for $p'_{\text{effect|cause}}$ is the best choice. Adapting the new p_{cause} while maintaining the same $p_{\text{effect|cause}}$ is known as **covariance shift**.

Note that this assumes that the causal graph has been correctly created during the learning phase of the initial tasks. Otherwise, this is not a safe assumption (Schölkopf et al. (2012) [158]).

In the case of a causal system with many variables the above means that modules that don't change when distribution changes don't have to be learned again. The model can use them as prior knowledge while learning the modules with new distributions.

4.6.7 Learning Independent Causal Mechanisms

Using relevant, modular and reusable mechanisms saves training time and minimizes the quantity of needed data for learning the changed modules. Modern neural networks are good at learning patterns from large independent and identically distributed (i.i.d) datasets, but are currently unable to effectively factorize their knowledge into reusable independent modules. A question naturally arises: how can one learn a causal model when no changes in the structure of the underlying model have occurred. Peters et al. (2018) [19] present an argument analogous to the Occam's Razor assumption (chapter 2.4.5) in the case of statistical learning. They suggest that if relatively simple mechanisms that explain the data at hand are found, then one be relatively certain of having discovered the correct causal model, or at least, a big part of it. They argue that the mechanisms that

result from the anti-causal factorization are usually much more complex than the true independent modular ones.

The next questions that needs to be addressed is how to learn these mechanisms using NNs, which have the advantage over other techniques of being able to generalize to unseen instances generated by the same distributions on which they were trained. Parascandolo et al. (2017) [20] presented a model consisting of **competing agents** that manage to learn independent causal mechanisms from purely observational data. During training, a sample may be generated from any of the independent mechanisms. The experts compete for the sample and only the winner is trained on inverting the transformation applied to the sample by the mechanism. The intuition behind employing competition is that the winning expert becomes further specialized in the corresponding mechanism. Moreover, since the mechanisms are independent, this generally does not improve the expert's performance on inverting the transformation generated by any other mechanism.

This is an unsupervised learning task on two levels. First, the original sample, before it is transformed by the mechanism, is not provided to the experts. Second and most important, the experts don't know which mechanism transformed the sample, which is what one would also expect when trying to train a model to solve a real-life task. The specialized expert is called to learn and recognize the patterns in the outputs of the associated mechanism.

Data

The training data consists of transformed and original MNIST digits [159]. The MNIST dataset is a well known computer vision classification dataset with images of hand-written digits along with the corresponding labels. Parascandolo et al. (2018) [20] apply 10 transformation to the digit images; 8 translations towards 8 different directions, contrast inversion and noise addition. These transformations represent 10 independent causal mechanisms. An original (canonical) image digit as well as transformed example are shown in figure 4.13.

The MNIST dataet is split in half. The transformations are randomly applied to one half. This ensures that the original and the transformed image will not be available simultaneously.

Models and Training

10 expert CNNs are called to learn the inverse functions of these transformations, each one with its own parameters, θ_i .

During training, a transform image $\mathbf{x}_{\text{tr}}$ is fed to all 10 experts. Each expert i, $i \in \{1, \cdots, 10\}$ produces its own proposition for the what the original image, $E_i(\mathbf{x}_{\text{tr}})$, could look like. The goal of the expert is to maximize an objective function $c : \mathbb{R}^{D_{in}} \to \mathbb{R}$ that receives high values in the space of images from the original dataset.

The function c is implemented using another CNN with its own parameters, θ_D. This CNN D is called a **discriminator** and the experts are called **generators**. Each one of the experts feeds the discriminator with its own suggestion and the latter decides which one made the most realistic proposal based on its own corresponding output values.

Only the parameters of the winning expert j are updated to maximize $E_j(\mathbf{x}_{\text{tr}})$, while the parameters of the other experts are left unaffected. The discriminator is updated to discern between the propositions of the winning experts and the original images. Therefore it is trained in identifying the original MNIST images of the second half of the dataset as such. This process is depicted in figure 4.13. Thus, by training the specialized experts to become even better at what they do the researchers attempt to fool the discriminator into thinking that their output images come from the original MNIST dataset. This type of model configuration, in the case of single generator, is called a **generative adversarial network (GAN)** and was introduced by Goodfellow et al. (2014) [160].

However, to avoid favouring the other experts, Parascandolo et al. (2018) [20] train their discriminator network, not only against the winning expert's output for a certain transformed image, but also against the propositions of the rest of the experts for the same image. The CE loss function of the discriminator therefore is equal to:

$$\max_{\theta_D}(E_{\mathbf{x}\sim P}\log(D_{\theta_D}(\mathbf{x})) + \frac{1}{10}\sum_{i=1}^{10} E_{\mathbf{x}_{tr}\sim Q}(\log(1 - D_{\theta_D}(E_i(\mathbf{x}_{tr}))))) \tag{4.9}$$

where P is the distribution of the original images and Q of the transformed images.

Parascandolo et al. (2018) explain that randomly initializing all experts before training does not work. The expert with the best initialization ends up winning all of the mechanisms. Therefore, after the random initialization step, all experts are pre-trained on identical input-output pairs randomly selected from the transformed part of the dataset. This renders them approximately equally capable when the main training stage begins.

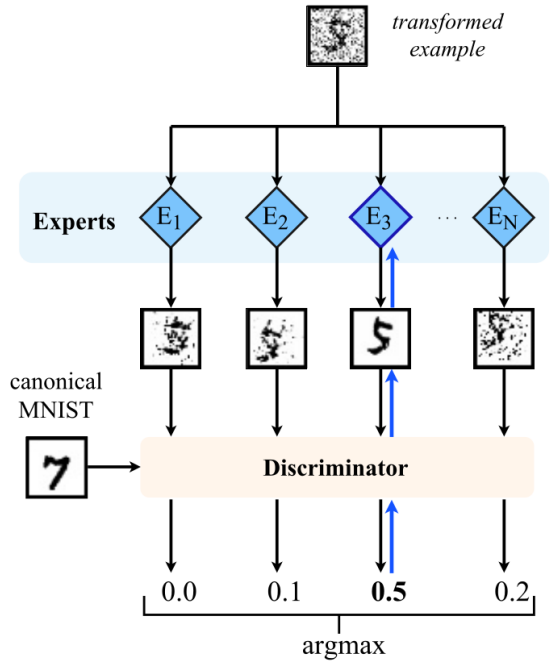


Figure 4.13. *The training process of the winning expert and the discriminator. The discriminator is trained using both the suggestions of the experts and the original MNIST digits. The winning expert is trained to improve his suggestions. The parameters of the experts that have lost are not updated [20].*

Results

The experiment was ran 10 times and the experts successfully specialized in 7 of them, with each expert learning a unique inverse transformation. The results of the rest 3 experiments were good but not perfect, as an expert might specialize in 2 distinct mechanisms while another one might not learn anything at all. The evolution of scores assigned to the experts by the discriminator during the training process in one of the seven successful runs are shown in figure 4.15. After a period of adjustment, the experts seem to successfully specialize in a distinct transformation each.

To test the quality of the inverse transformations implemented by the experts their results were fed to a pre-trained MNIST classifier as inputs. This classifier achieves 99% accuracy on the original dataset. It is shown in figure 4.15 that the experts' outputs are correctly classified once they are fully trained. On the other hand, when the transformed images are fed to the classifier, without having first applied the inverse transformations, it only achieves a 40% accuracy, as indicated by the starting point of the figure 4.15.

They also tried training a single big model to learn all the transformations. This model has over twice the combined number of parameters of all the experts but still failed to learn the transformations. This is an excellent example of how the incorporation of prior knowledge in a model's architecture (chapter 2.5.2) can play a crucial role in its success.

Because of the small size and thus the simplicity of these experts, they learn invariant features related to the corresponding transformations. They are thus able to generalize to a different dataset of images of written characters from 50 different alphabets, called Omniglot [161] Furthermore,

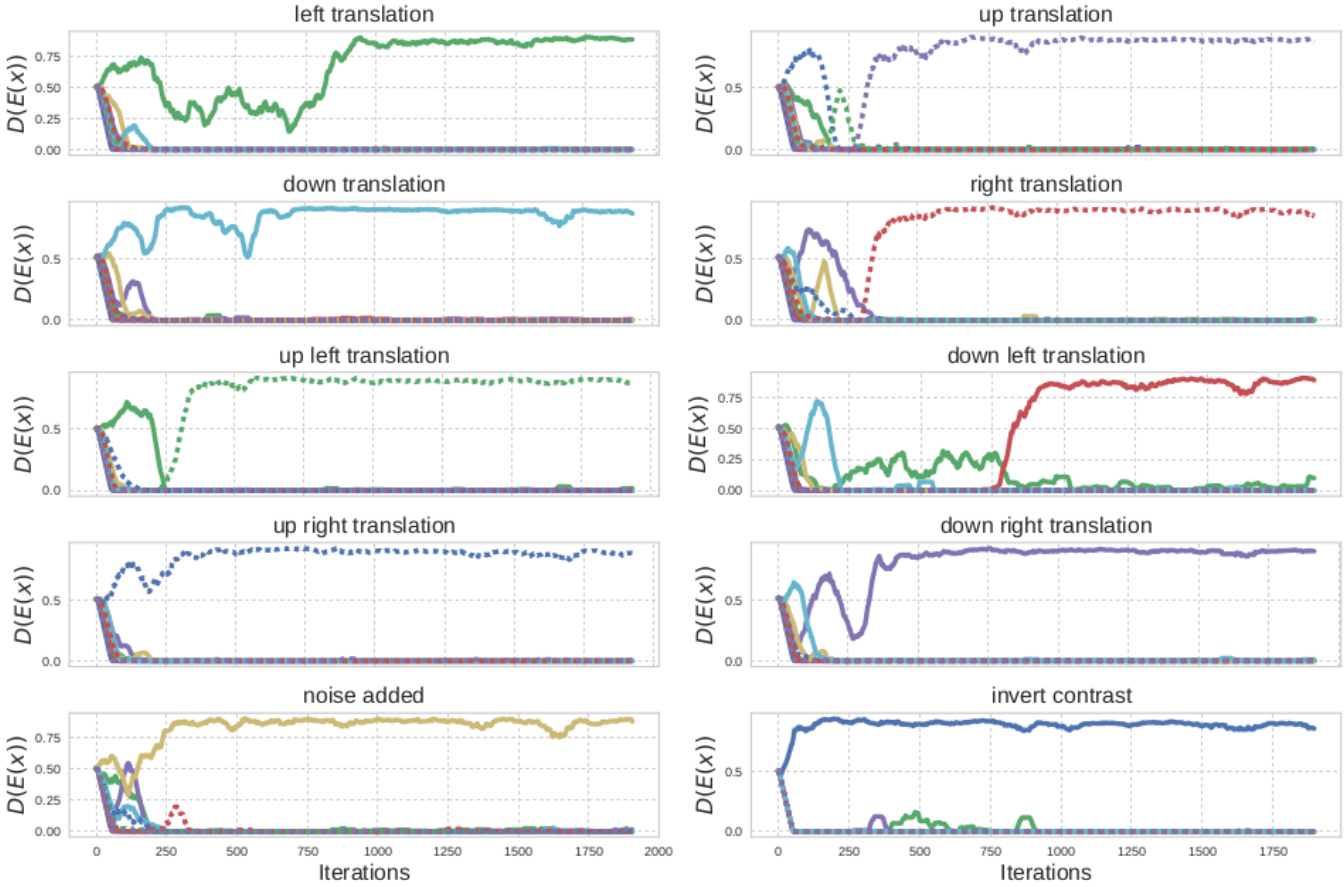


Figure 4.14. *The evolution of the scores assign by the discriminator to each expert during a successful run [20].*

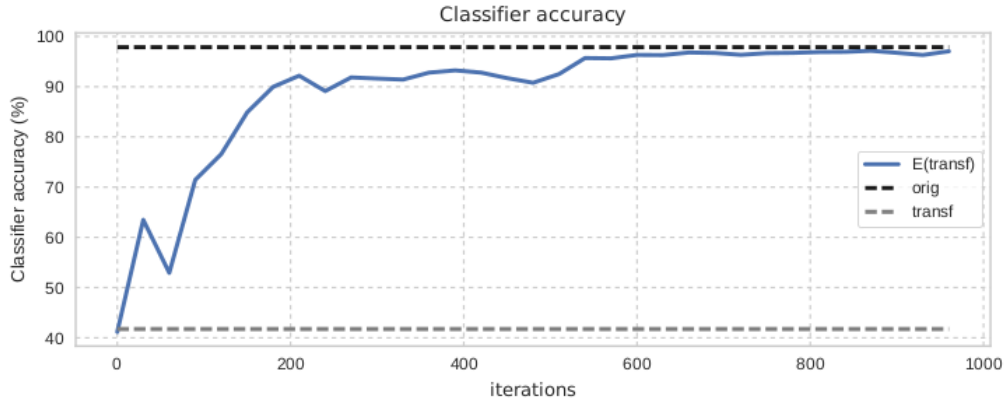


Figure 4.15. *The evolution of the MNIST classifier's accuracy on the images transformed by a mechanism and then inversely transformed by an expert during the experts' training [20].*

two or more experts can be serially combined to implement a complex transformation as shown in figure 4.16b.

Number of Mechanisms

An underlying assumption adopted so far is that the number of mechanisms is known. Yet, it is generally not known a priori. Parascandolo et al. (2018) tested what happens in the case where the mechanisms are assumed to be more than they truly are. They found that, in that case, some

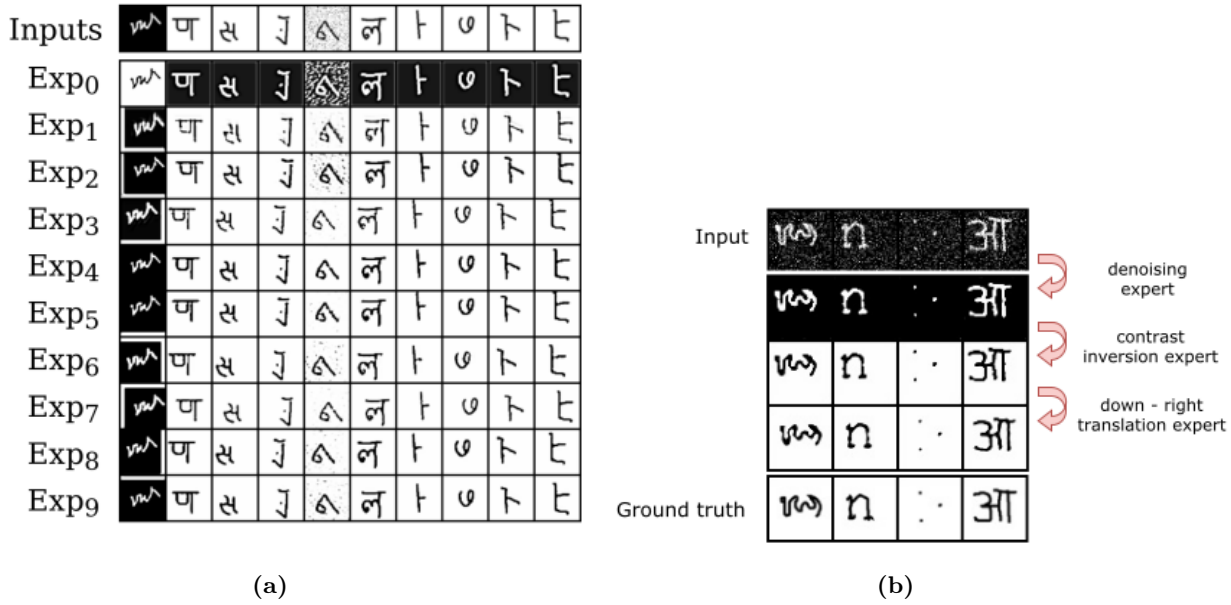


Figure 4.16. ***(a)*** *The 10 experts applied to 10 Omniglot characters transformed in all possible ways. The propositions for the original images lie across the diagonal* ***(b)*** *Serial use of different experts on a set of characters [20]*

experts do not specialize at all and some inverse transformations are divided between two experts, with each one focusing on a distinct part of the transformed dataset. The solution they propose is pruning the ones that are not used and merging the ones that learn to perform versions of the same transformation.

If, on the other hand, fewer mechanisms are assumed to make up the underlying model than they truly do, some experts learn more than one inverse transformation and some others are not learned by any expert.

4.7 Conditional Computation

Before delving into the matter of conditional computation, it is useful to clarify some common misconceptions regarding the difference between two learning approaches that involve the learning of more than one tasks, multi-task learning and transfer learning. Transfer learning was also discussed in chapter 3.9.2, but a more thorough analysis of the matter is needed.

When there more than one tasks that must be dealt with, $t_1, \cdots, t_T$, one may choose a **single-task learning** approach, i.e. to learn each task independently by training T separate models. But in doing this, one fails to take advantage of the similarities between these tasks and learn features that could be used by multiple tasks. This is especially useful in cases where there is a shortage of data related to one or more tasks, and tasks exhibit structural similarities between each other. Moreover, single-tasks learning leads to a rapid increase in the number of parameters w.r.t. the number of tasks and training from scratch is usually much more time consuming than using already learned features.

Multi-task and transfer learning approaches both attempt to benefit from the commonalities between the various tasks that are learned by the models.

4.7.1 Multi-task Learning

Multi-task learning is an approach that aims to learn a set of tasks $t_1, \cdots, t_T$ simultaneously. This happens by imposing constraints on the relation of parameters that store knowledge related to different tasks. These constraints remain active throughout the joint optimization of the loss functions of the T tasks and thus affect the evolution of the parameter values.

The most common type of constraint is called **hard parameter sharing** [162]. Parameter sharing has already been introduced as a mechanism that implements a belief that some variables should be processed in similar ways (chapter 2.5.2). By applying it to solve multiple tasks, one imposes the constraint that parts of the models that are used to solve different tasks will be unified into a single neural structure. For example, one may choose to use the same first layers for all tasks, which will act as feature extractors, and then use task-specific neural modules on top of them. This is depicted in figure 4.17.

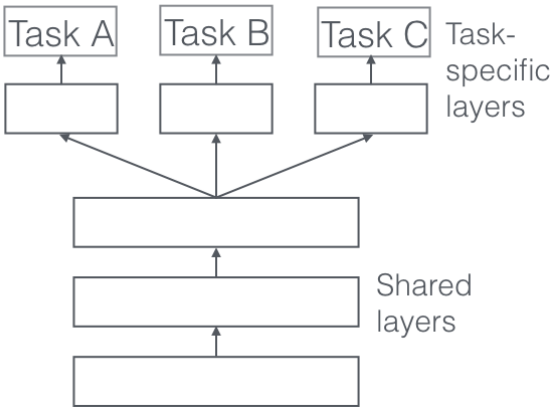


Figure 4.17. *Example of hard parameter sharing in a neural model that is trained on three tasks simultaneously. Figure from* `https://avivnavon.github.io/blog/parameter-sharing-in-deep-learning/`.

Similarly, one can use a **soft parameter sharing** mechanism that controls how much the parameters that are used to handle different tasks are allowed to differ. This option allows for more flexible models but leads to the increase of the number of parameters and demands that the rule controlling the way the parameters are linked is properly tuned. One could, for example, use the euclidean distance as a metric for estimating parameter distance.

Researchers view multi-task learning as a form of **inductive bias**, guiding the model into selecting some hypotheses and ignoring others. The aforementioned mechanisms reduce the degrees of freedom of the model and, like the weight sharing prior, effectively mitigate the problem of overfitting. The features that are learned by the shared parameters are bound to be more general than they would otherwise be, as they must meet the needs of many tasks. Overfitting is also related to the noise in the training data, as the model ends up learning the noise and not the underlying mathematical structure. But datasets belonging to different tasks usually contain independent noise which means that the shared parameters are likely to average these task-specific noise components out.

4.7.2 Transfer Learning

Transfer learning (Pratt et al. (1991) pratt1991direct, Pratt (1993) [163]) aims at employing knowledge acquired from the training a set of tasks $t_1, \cdots, t_{T-1}$, called **source tasks** to effectively learn a new task t_T, called a **target task**. Like in the case of multi-task learning a prerequisite for successful transfer learning is the existence of similarities between the source and the target tasks that leads to the learning of features that can be used by all of them. In most transfer learning applications it is true that $T = 2$ and one seeks to take advantage of the data abundance of the source tasks(s) to be able to quickly improve at the target task that is usually accompanied by a much smaller dataset.

The main difference between multi-task and transfer learning is that, while multi-task learning seeks to improve the model's capabilities at performing all T tasks, transfer learning's only goal

is to use knowledge gathered from the source tasks to enable the model to improve at the target task, without necessarily caring if the model's performance on the source tasks deteriorates as a result.

In a transfer learning setting, one usually employs a model trained on the source tasks to also learn how to perform the target task. One may choose to train a new model from scratch on the source tasks or use an already pre-trained model (chapter 3.9). Then it is possible to further tune the entire or part of the model to the target task (chapter 3.9.2), or use it as it is to perform the task. Since it is possible for the source and target tasks to use a different set of labels for their examples, randomly initialized task-specific neural structures are usually installed on top of the pre-trained model and trained on the target task.

An example of use of the transfer learning methodology is the pre-training of Transformer-based language models, followed by their fine-tuning on the downstream tasks, like in the case of the BERT model, discussed in chapter 3.9.3. The use of embeddings (chapter 3.4), sparse and pre-trained dense ones, in new tasks is another example. Furthermore, pre-trained CNN features on large computer-vision datasets are known to be very effective at improving model performance when data is scarce [164].

An interesting line of research involves a set of methods that selectively train parts of a neural network faster than others. Significantly, despite their many commonalities, not all of them were designed to serve the same purpose.

4.7.3 Freezing Layers

Layer freezing is a common approach used in transfer learning (chapter 4.7.2). Fine-tuning a pre-trained model on a task with a big dataset usually ensures that it improves at it and performs well on its test set. Yet, a large pre-trained model is fine-tuned on a downstream task with low data availability, according to chapter 2.4.5, is known to overfit. Then, valuable knowledge store in its parameters during pre-training might be lost and it the model may not be able to generalize to the downstream task's test set.

A method that is usually employed to prevent this is **freezing** the lower layers of the pre-trained model while fine-tuning the rest. This means that gradients are not computed for the frozen parameters, which are then not updated at all during fine-tuning but retain the values the acquired during pre-training. The parameters of the upper layers are trained according to standard procedure. The reason of freezing the lower layers only is because these layers are known to learn basic features which are usually useful during for both the pre-training and the downstream tasks, and should thus not be altered. Upper layers are known to learn task-specific features and must therefore be retrained.

Typically, as the number of available training samples for the downstream task increases, one is able to unfreeze more layers starting from the one closest to the output and continuing by sequentially unfreezing layers towards the input one. The more layers are trained usually the better the model performs on the downstream task's training set. On the other hand, keeping more layers frozen reduces data requirements and training time, since fewer computations need to be performed.

4.7.4 Gating

The matter of **gating** was briefly discussed in chapter 2.6.6, as gates are used by LSTMs to choose which pieces of knowledge will be granted passage to the next computational stage and

which will not. Gates in LSTMs are single-layer NNs with sigmoid functions attached to their outputs that perform the aforementioned decision for every dimension of a vector signal.

Srivastava et al. (2015) [165] for example, also implement gates to balance the use of residual connections with the signals coming from the intermediate layer. Essentially, layers can be viewed as a king of memory that is accessed whenever allowed by the corresponding gates.

4.7.5 Conditional Computation

Bengio (2013) [166] proposes **conditional computation** as a means to meet the computational requirement of building extremely big models. He identifies the linear scaling of the computations performed by a NN during inference w.r.t. the number of its parameters as one of the main problems. By applying computation the NN learns to employ only part of its units when it propagates forward and deactivates the rest, whose outputs are deemed to be irrelevant. A variant of condition is employed by a famous machine learning model, decision trees [167], that perform sequential decisions, depending on the input, concerning the subgroups in which it belongs.

Bengio (2013) [166] proposes combining sparse activations and multiplicative connections to implement conditional computation. The **sparse activation** method enables only few units to activate while the rest are deactivated, and thus are neither considered during computations (forward propagation) nor are they updated (backpropagation) while they remain that way. This can be implemented, for example, by including a $L1$ regularization term on the number of active units in the loss function. **Multiplicative connections** act as gates; some units gate other units. If the first set of units is sparsely activated then, through the multiplicative connections, it forces the same pattern on the second set of units, deactivating all but a few of them.

Decision trees differ in that the choice of following a path automatically excludes the choice of following other paths starting from the same node as well as their children. On the other hand, activating a NN unit should not, according to Bengio (2013) [166], exclude any other from activating too. This preserves the advantage of distributed representations held by NNs, while also reducing the computational burden.

A problem that similar methods face is that some gating units tend to remain deactivated during the entire training process, and thus the related resources are never utilized. In order to produce training signals for the gating units, Bengio (2013) [166] proposes the introduction of randomness in the activation of gates, as he believes it will force some gates that would otherwise not be trained to have their parameters updated.

4.8 Meta-Learning

4.8.1 Motivating Meta-Learning

As Vinyals et al. (2016) [168] explains, in a machine learning problem train and test conditions must match. It is known that the real reason of training a model is not so that it performs well on the training set, but in order to improve its generalizing skills on the test set. Training data is only one of the tools required to achieve this.

Therefore, instead of training models on source tasks, hoping that pre-trained models will serve as good initialization points when one transitions to a target task, one could explicitly train the models on the source tasks to optimize their ability to generalize to test data. Unavoidably, the assumption that source and target tasks are sampled from a common task distribution $\mathcal{T}$ has to be made. But, if that is the case, then it is very reasonable to expect that the models will able to generalize to the test data of the target tasks after they are trained to do so to the test data of the source tasks.

This intuition is implemented by a learning framework called **meta-learning**, which essentially means *learning how to learn* (Schmidhuber (1987) [169], Bengio et al. (1990) [170]). Meta-learning methods are usually employed in low-data regimes, and the goal is to create a model that is able to generalizing to test data even when it is given few training instances. In contrast to transfer learning, meta-learning assumes the existence of many source tasks, each one with its own dataset $D_m = \{(x_m^{(1)}, y_m^{(1)}), \cdots, (x_m^{(T_m)}, y_m^{(T_m)})\}, m \in \{1, \cdots, M\}$. D_m is split into a training set, $D_{m,tr} = \{(x_{m,tr}^{(1)}, y_{m,tr}^{(1)}), \cdots, (x_{m,tr}^{(T_{m,tr})}, y_{m,tr}^{(T_{m,tr})})$, and a test set, $D_{m,ts} = \{(x_{m,ts}^{(1)}, y_{m,ts}^{(1)}), \cdots, (x_{m,ts}^{(T_{m,ts})}, y_{m,ts}^{(T_{m,ts})})$. Each task is seen as a training sample, and, since train and test conditions must match, each source task usually comes with a training set of size approximately equal to size of the training set of the target task(s) [21].

4.8.2 Probabilistic View

The goal of the standard supervised learning approach that has been discussed thus far is to estimate the optimal model parameters:

$$\hat{\boldsymbol{\vartheta}} = arg\max_{\boldsymbol{\vartheta}} \log p(\boldsymbol{\vartheta}|D) = arg\max_{\boldsymbol{\vartheta}} \log p(D|\boldsymbol{\vartheta}) + \log p(\boldsymbol{\vartheta}) \tag{4.10}$$

which essentially is equation 2.6.

In meta-learning, one seeks to employ additional tasks with datasets $D_{mt} = \{D_1, \cdots, D_M\}$ to learn the model's parameters:

$$\hat{\boldsymbol{\vartheta}} = arg\max_{\boldsymbol{\vartheta}} \log p(\boldsymbol{\vartheta}|D, D_{mt}) \tag{4.11}$$

What usually happens is that knowledge from D_{mt} is incorporated into meta-parameters $\boldsymbol{\varphi}$:

$$\begin{aligned} \log p(\boldsymbol{\vartheta}|D, D_{mt}) &= \log \int_{\boldsymbol{\varphi}} p(\boldsymbol{\vartheta}|D, \boldsymbol{\varphi}) p(\boldsymbol{\varphi}|D_{mt}) d\boldsymbol{\varphi} \\ &\approx \log p(\boldsymbol{\vartheta}|D, \hat{\boldsymbol{\varphi}}) + \log p(\hat{\boldsymbol{\varphi}}|D_{mt}) \end{aligned} \tag{4.12}$$

In the first equality it is assumed that $\boldsymbol{\vartheta}$ is independent of D_{mt} given $\boldsymbol{\varphi}$, which is a natural conclusion since all knowledge from D_{mt} is transferred to $\boldsymbol{\varphi}$. The second approximate equality is a common assumption made in similar situations, where it is assumed that the mass of the probability distribution is gathered around the optimal value, and therefore replacing with this value results in a good approximation.

Equation 4.12 splits the problem into a **meta-learning problem**:

$$\hat{\boldsymbol{\varphi}} = arg\max_{\boldsymbol{\varphi}} \log p(\boldsymbol{\varphi}|D_{mt}) \tag{4.13}$$

and an **adaptation problem**:

$$\hat{\boldsymbol{\vartheta}} = arg\max_{\boldsymbol{\vartheta}} \log p(\boldsymbol{\vartheta}|D, \hat{\boldsymbol{\varphi}}) = f_{\hat{\boldsymbol{\varphi}}}(D) \tag{4.14}$$

4.8.3 Meta-Learning Process Overview

When trying to design a meta-learning algorithm, one first has to specify the form of $f_{\hat{\boldsymbol{\varphi}}}$. Then one chooses how to find $\hat{\boldsymbol{\varphi}}$ by transferring knowledge from D_{mt}. The phase of estimating $\hat{\boldsymbol{\varphi}}$ is called the **meta-training phase**. Every time a new task T_i is presented, $f_{\hat{\boldsymbol{\varphi}}}$, along with $D_{i,tr}$, are used to generate $\boldsymbol{\vartheta}_i = f_{\hat{\boldsymbol{\varphi}}}(D_{i,tr})$. The model parameterized by $\boldsymbol{\vartheta}_i$ then produces predictions for the features of the respective test set's samples $D_{i,ts}$. The loss computed on the test set of T_i is

used to train $\boldsymbol{\varphi}$ through its contribution to the choice of $\boldsymbol{\vartheta}_i$. This process is shown in figure 4.18a. After $\hat{\boldsymbol{\varphi}}$ is estimated, the model is shown the training data of the target task, D_{tr}, and outputs:

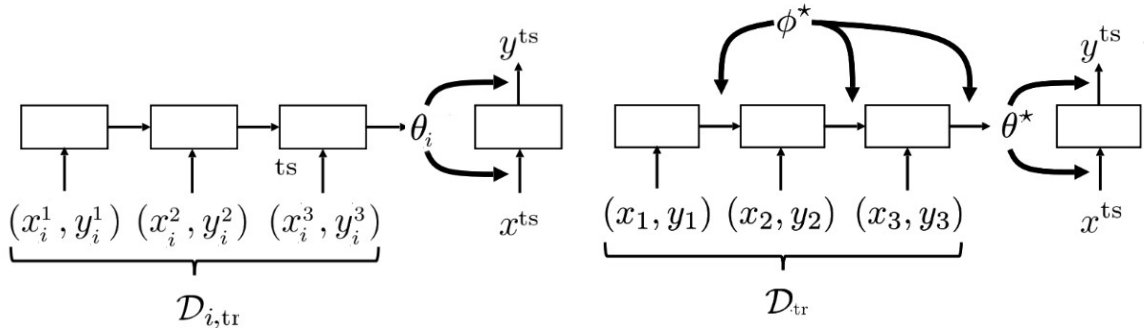


Figure 4.18. ***(a)*** *Meta-learning training process. First estimate $\boldsymbol{\vartheta}_i$, given $D_{i,tr}$ and $\boldsymbol{\varphi}$. Then, the model, parameterized by $\boldsymbol{\vartheta}_i$, makes predictions for the task's test set $D_{i,ts}$. The respective gradients are used to update $\boldsymbol{\varphi}$.* ***(b)*** *Meta-learning testing process. First estimate $\boldsymbol{\vartheta}$, given D_{tr} and $\boldsymbol{\varphi}$. Then, the model, parameterized by $\boldsymbol{\vartheta}$, makes predictions for the task's test set D_{ts} [21].*

$$\hat{\boldsymbol{\vartheta}} = arg\max_{\theta} \log p(\theta | D_{tr}, \hat{\boldsymbol{\varphi}}) = f_{\hat{\boldsymbol{\varphi}}_i}(D_{tr}) \tag{4.15}$$

Then it is shown D_{ts} and uses $\hat{\boldsymbol{\vartheta}}$ to make its predictions. This phase is called the **meta-testing phase** and is shown in figure 4.18b.

The problems of hyper-parameter optimization and architecture search are actually examples of meta-learning problems. The hyper-parameters or the architecture play the role of the meta-parameters, $\boldsymbol{\varphi}$, and the network weights the role of the adaptation parameters, $\boldsymbol{\vartheta}$. Another problem that is considered by many to be a meta-learning problem is **few-shot learning**, that will be discussed in chapter 4.8.4.

4.8.4 Few-Shot Learning

If a child is shown a grown up elephant and a baby elephant, then after seeing another animal, like a giraffe, it is quick to recognize its babies even if it has never seen a baby giraffe before. Humans are able to perform valid generalizations while utilizing very few examples. Neural network training unfortunately, and especially deep learning, demands large and expensive datasets.

Few-shot learning is a learning framework that attempts to enable the training of neural networks with very few examples. The datasets of classification problems of such nature that are categorized as being N-way k-shot learning problems contain examples from N different classes, and k examples are available for each class. The special case of 0-shot learning problems consists of tasks in which the classifier is only provided with a high-level description of each class, without being shown any training samples.

Many researchers classify few-shot learning as a meta-learning problem. Many of the recently proposed approaches to few-shot learning make use of a variety of training tasks, handling each task as a separate sample in the same sense tasks are used by meta-learning algorithms. In that sense, to ensure uniform train and test conditions, the training datasets of meta-training tasks must contain a number of examples that is comparable to the one found in the training set of the meta-testing tasks [171].

But, even though few-shot learning has been described as a meta-learning problem, that does necessarily mean that one is obliged to follow the meta-learning framework when trying to few-shot learning. Brown et al. (2020) [22] train that a massive 175 billion parameter Transformer-based model on a huge language modelling dataset based on [134], named **GPT-3**. They prove that it can perform few-shot, one-shot and zero-shot learning without ever being fine-tuned on the downstream tasks. Training samples are instead placed in the context window of the decoder-like model and serve as indicators of the tasks that needs to be performed in one-shot and few-shot problems. Typically 10-100 input/output example pairs fit inside this window. A high-level task description is used as context in zero-shot problems.

They prove that few-shot performance improves with model size (figure 4.19). In fact, GPT-3 competes with and even outperforms many pre-trained and fine-tuned sota models and strong baselines in several NLP problems, like QA, common-sense reasoning, reading comprehension and on the SuperGLUE [172] dataset in the context of zero, one and few-shot learning.

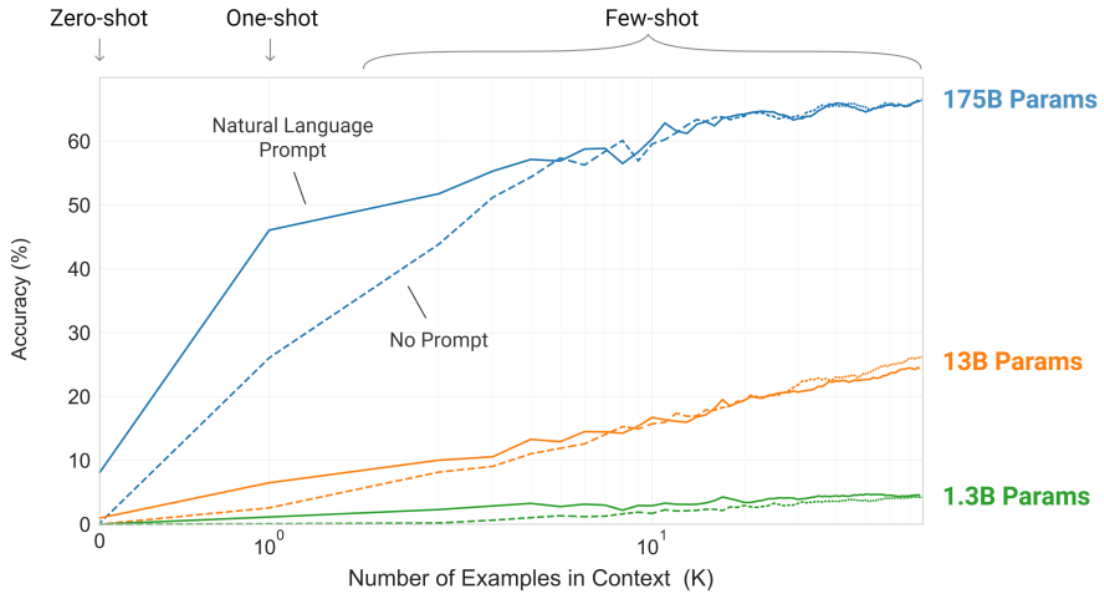


Figure 4.19. *Performance on a simple task requiring the model to remove random symbols from a word, both with and without a natural language task description. The steeper the curve the better a model becomes as the number of in-context examples increases. Larger models thus make better use of examples than smaller models. They report seeing this behaviours in a variety of tasks [22].*

4.8.5 Black-Box Adaptation

There are several approaches to meta-learning. **Black-box adaptation** approaches use neural networks to produce a deterministic estimate of the task-specific parameters θ_i for a task T_i, $\varphi = f_\varphi(D_{i,tr})$. The used NNs must be able to handle a set of feature-label pairs as input, which belong to $D_{i,tr}$. Architectures like RNNs, 1-dimensional CNNs, or self-attention networks may thus be used. These models are parameterized by ϕ, which is then updated using the error signals produced by the predictions of a neural model g_{ϑ_i} for the test sample of task T_i, $D_{i,ts}$:

$$\max_{\varphi} \sum_{T_i} \sum_{(x,y)\sim D_{i,ts}} \log g_{\vartheta_i}(y|x) = \max_{\varphi} \sum_{T_i} L(f_\varphi(D_{i,tr}), D_{i,ts}) \tag{4.16}$$

The way different tasks are actually treated as samples by meta-learning algorithms now becomes obvious. The black-box adaptation algorithm is thus the following:

Algorithm 4.1: *Black-Box Adaptation*

initialize $\boldsymbol{\vartheta}$ and $\boldsymbol{\varphi}$
repeat
  Sample $T_i \sim \mathcal{T}$
  Sample disjoint sets $D_{i,tr}$, $D_{i,ts} \sim \mathcal{D}_i$
  $\boldsymbol{\vartheta}_i = f_{\boldsymbol{\varphi}}(D_{i,tr})$
  $\boldsymbol{\varphi} := \boldsymbol{\varphi} - a\nabla_{\boldsymbol{\varphi}} L(\boldsymbol{\vartheta}_i, D_{i,ts})$
until convergence

Unfortunately, this method, as is described, does not scale well with increasing dimensions of $\boldsymbol{\vartheta}$, i.e. for big models. Instead of computing $\boldsymbol{\vartheta}$, one can alternatively choose to have f outputting a low dimensional vector $\mathbf{h}$, that incorporates the statistical information present in $D_{i,tr}$. Then, $\boldsymbol{\vartheta}$ is created by a model parameterized by the meta-learned parameters $\boldsymbol{\varphi}_g$, which uses $\mathbf{h}$ as input.

4.8.6 Optimization-Based Approach

Another possible approach is to employ an **optimization-based procedure** for producing the adaptation parameters $\boldsymbol{\vartheta}_i$. The meta-learning parameters can play the role of priors in this process. One of the most successful ways of incorporating priors in deep learning is through the initialization process. This is the case with pre-training and fine-tuning, for example, where prior knowledge about natural language is distilled into the pre-trained model's parameters that serve as an initialization point for the fine-tuning process.

The idea of Finn et al. (2017) [23] was to to use the meta-parameters $\boldsymbol{\varphi}$ as an initialization to a model that is trained on the train sets of the various tasks with a standard GD process through which $\boldsymbol{\vartheta}$ is computed. They aim at meta-learning initializations that enable models to generalize well even in a few-shot learning scenario. They explain that the discovered initializations increase the model's sensitivity to training so that training, even with the use of few training samples provided by any task $T_i \sim \mathcal{T}$, leads to an area of the parameter space that corresponds to low generalization error for that particular task. This is elegantly shown in the figure 4.20 provided by Finn et al. (2017) [23].

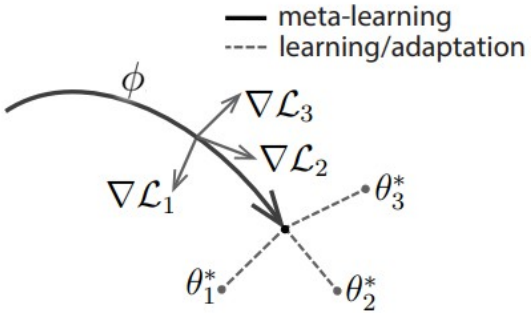


Figure 4.20. *During the meta-learning process the model's initialization which coincides with the meta-parameters is trained. The goal is for the corresponding parameter vector, $\boldsymbol{\varphi}$, to reach a point in the parameter space that, when used as an initialization, enables the model to quickly learn parameter values that ensure good generalization [23].*

Mathematically, the training the adaptation parameters is denoted as:

$$\boldsymbol{\vartheta}_i \leftarrow \boldsymbol{\varphi} - a\nabla_{\boldsymbol{\varphi}} L(\boldsymbol{\varphi}, D_{i,tr}) \tag{4.17}$$

Then the meta-learning problem is formulated as:

$$\min_{\boldsymbol{\varphi}} \sum_{T_i} L(\boldsymbol{\varphi} - a\nabla_{\boldsymbol{\varphi}} L(\boldsymbol{\varphi}, D_{i,tr}), D_{i,ts}) \tag{4.18}$$

since the error signals on the test set are used to produce gradients for the training of the meta-parameters $\boldsymbol{\varphi}$. The respective algorithm, 4.2, is the same as alg. 4.1 after changing the fifth line. This technique is called **model-agnostic meta-learning (MAML)** since, because of the use of GD, it is agnostic to the specific model architecture and thus can utilize any model that can be trained with GD.

ALGORITHM 4.2: *Optimization-Based Approach*

initialize $\boldsymbol{\vartheta}$ and $\boldsymbol{\varphi}$
repeat
  Sample $T_i \sim \mathcal{T}$
  Sample disjoint sets $D_{i,tr}$, $D_{i,ts} \sim \mathcal{D}_i$
  $\boldsymbol{\vartheta}_i = \boldsymbol{\varphi} - a_{in} \nabla_{\boldsymbol{\varphi}} L(\boldsymbol{\varphi}, D_{i,tr})$
  $\boldsymbol{\varphi} := \boldsymbol{\varphi} - a_{out} \nabla_{\boldsymbol{\varphi}} L(\boldsymbol{\vartheta}_i, D_{i,ts})$
until convergence

It also has the useful inductive bias of computing model parameters using the well tested process of GD. Finn and Levine (2018) [21] found that for sufficiently deep networks, MAML can approximate any function of $D_{i,tr}, x_{ts}$ and is therefore it is equally powerful to black-box adaptation methods.

4.8.7 Non-Parametric Methods

All models presented in chapters 2 and 3 are **parametric**, meaning they have a set of parameters that are updated during training to fit a training set. **Non-parametric methods** on the other hand don't use parameters to perform inference but rely on the actual samples of the training set.

Non-parametric methods are known to perform well in few-shot learning settings. But, in the case of meta-learning, it is assumed that a large number of tasks is usually available, even though the data related to each one of them might be scarce. What is needed is an approach that benefits from the effectiveness of non-parametric methods in few-shot learning problems while also putting the abundance of different tasks to good use.

A commonly used method is to employ parametric networks to learn during meta-training an embedding space, in which non-parametric approaches can be applied. Vinyals et al. (2016) [168], for example, train a NN, parameterized by θ, to create embeddings of the training data of each task and, along with the help of an attention mechanism applied on them, use these embeddings to classify test samples.

Snell et al. (2017) [171] compute the mean of the embeddings of all training samples, S_k, belonging to the same class k, and use the resulting embedding as the class prototype, $\mathbf{C}_k = \frac{1}{|S_k|} \sum_{(x^{(i)}, y^{(i)}) \in S_k} f_\theta(x^{(i)})$. To classify a new point they find the class prototype that is closer to the point's embedding, as measured by a euclidean distance metric. For 0-shot problems they embed the high-level description of the class itself instead of samples.

4.8.8 Few-Shot Learning NLP Tasks

Bansal et al. (2020) [173] mix some of the aforementioned notions in an effort to train a model to few-shot learn NLP tasks. **LEOPARD** (learning to generate softmax parameters for diverse classification), the method they develop, is an optimization-based meta-learning approach that attempts to solve the problem of disjoint sets of disjoint sets of labels among different tasks faced by MAML, while retaining the ability to benefit from larger training sets.

In the core of the model is the 12-layer BERT model (chapter 3.9.3), parameterized by $\boldsymbol{\vartheta} = \{\boldsymbol{\vartheta}_1, \cdots, \boldsymbol{\vartheta}_{12}\}$. It accepts input sentences as its input and produces D_{model}-dim contextual embeddings, $\tilde{\mathbf{x}} = f_{\boldsymbol{\vartheta}}(\mathbf{x})$. To enable the model to handle tasks with different numbers of classes they use the contextual embeddings to create task-dependent weight and bias parameters for a softmax layer. Similarly to Snell et al. (2017) [171], for a task T_i, they generate distinct task-specific parameters for each class based on the sets of available training samples that belong to it, $C_i^n = \{x_j | y_j = n\}$, $n \in [N_i]$. For the n-th class among $|C_i|$ classes, $\mathbf{w}_i^n$ and b_i^n are generated by:

$$\mathbf{w}_i^n, b_i^n = \frac{1}{|C_i^n|} \sum_{\mathbf{x}_j \in C_i^n} g_{\boldsymbol{\Psi}}(f_{\boldsymbol{\vartheta}}(\mathbf{x}_j)) \tag{4.19}$$

where $g_{\boldsymbol{\psi}}$ is a FFNN, parameterized by $\boldsymbol{\psi}$, $\mathbf{w}_i^n \in \mathbb{R}^{D_{soft}}$ and $b_i^n \in \mathbb{R}$. The concatenation of all task-specific parameters together is symbolized as $\mathbf{W}_i = [\mathbf{w}_i^1; \cdots; \mathbf{w}_i^{N_i}]$ and $\mathbf{b}_i = [b_i^1; \cdots; b_i^{N_i}]$. Then, for a new sample $\mathbf{x}$, it is true that:

$$p(y|\mathbf{x}) = \text{softmax}(\mathbf{W}_i h_{\boldsymbol{\varphi}}(f_{\boldsymbol{\vartheta}}(\mathbf{x})) + \mathbf{b}_i) \tag{4.20}$$

where $h_{\boldsymbol{\varphi}}$ is another FFNN parameterized by $\boldsymbol{\varphi}$. The softmax produces an output over the N_i classes of the task.

Since applying MAML to all 110m. BERT parameters may not be possible, they separate the parameters into task-specific and task-agnostic. **Task-agnostic parameters** include the parameters found at the u-th BERT layer and below it, denoted by $\boldsymbol{\vartheta}_{\leq u}$, as well as $\boldsymbol{\psi}$. They are symbolized as $\boldsymbol{\Psi} = \boldsymbol{\vartheta}_{\leq u} \cup \{\boldsymbol{\psi}\}$, where u is a hyper-parameter. These parameters are trained in a MAML-based optimization process as they must provide a good initialization points for learning other tasks. **Task-specific parameters** $\boldsymbol{\Theta}_i = \boldsymbol{\vartheta}_{>u} \cup \{\boldsymbol{\varphi}, \mathbf{W}_i, \mathbf{b}_i\}$. Task-specific parameters are adapted separately for each task.

To benefit from bigger training sets, they sample $G > 1$ disjoint training subsets for each task. The first one is used to create $\mathbf{W}_i$ and $\mathbf{b}_i$. The rest are used to update the task specific parameters:

$$\boldsymbol{\Theta}_i^{(j+1)} = \boldsymbol{\Theta}_i^{(j)} - a_j E_{D_{i,tr} \sim T_i}[\nabla_{\boldsymbol{\Theta}_i} L_i(\{\boldsymbol{\Psi}, \boldsymbol{\Theta}_i\}, D_{i,tr})] \tag{4.21}$$

in the inner loop. Task-agnostic parameters are updated in the outer loop. The algorithm is the following:

Algorithm 4.3: *LEOPARD*

Input: set of M training tasks and losses $(T_1, L_1), ..., (T_M, L_M)$, model parameters $\boldsymbol{\Psi} = \boldsymbol{\vartheta}$, $\boldsymbol{\psi}$, α, hyper-parameters u, G, β ▷ Meta-Learn a different learning rate for each layer
Initialize $\boldsymbol{\vartheta}$ with pre-trained $\text{BERT}_{\text{base}}$
repeat
  Sample batch of tasks
  for all $T_i \in T$: **do**
    Sample $D_{i,tr} \sim T_i$
    $C_i^n = \{x_j | y_j = n\}$
    $\mathbf{w}_i^n, b_i^n = \frac{1}{|C_i^n|} \sum_{\mathbf{x}_j \in C_i^n} g_{\boldsymbol{\psi}}(f_{\boldsymbol{\vartheta}}(\mathbf{x}_j))$
    $\mathbf{W}_i = [\mathbf{w}_i^1; \cdots; \mathbf{w}_i^{N_i}]$, $\mathbf{b}_i = [b_i^1; \cdots; b_i^{N_i}]$
    $\boldsymbol{\Theta}_i^{(0)} = \boldsymbol{\vartheta}_{>u} \cup \{\boldsymbol{\varphi}, \mathbf{W}_i, \mathbf{b}_i\}$
    for $j \in \{0, \cdots, G-1\}$ **do**
      Sample $D_{i,tr} \sim T_i$
      $\boldsymbol{\Theta}_i^{(j+1)} = \boldsymbol{\Theta}_i^{(j)} - a_j E_{D_{i,tr} \sim T_i}[\nabla_{\boldsymbol{\Theta}_i} L_i(\{\boldsymbol{\Psi}, \boldsymbol{\Theta}_i\}, D_{i,tr})]$
    end for
    Sample $D_{i,val} \sim T_i$
    $g_i \leftarrow \nabla_{\boldsymbol{\Psi}} L_i(\{\boldsymbol{\Psi}, \boldsymbol{\Theta}_i\}, D_{i,val})$
  end for
  $\boldsymbol{\Psi} := \boldsymbol{\Psi} - \beta \sum_i g_i$
until convergence

In order to stabilize training, they initialize LEOPARD from the pre-trained BERT.

Results

They evaluate on several NLP tasks, including some from the GLUE dataset. The model's parameters are trained on a set of training tasks and fine-tuned with k training examples per label for a target task, which are not seen during training.

The model generally performed better than a fine-tuned BERT and a BERT trained on various

multi-task learning tasks. It is also good at few-shot domain adaptation problems performing better than strong baselines.

The results indicate LEOPARD's ability to few-shot learn how to perform tasks with varying numbers of classes. It learns better parameter initializations for few-shot learning than a version of BERT that uses self-supervised pre-training and another one that employs pre-training followed by multi-task learning.

4.8.9 Continual Learning

Throughout their lives, humans learn to perform a variety of tasks. These tasks are not all presented to them simultaneously, like they are in the multi-task setting, but sequentially. To solve a new task, they apply knowledge acquired from solving previous tasks as has already been discussed. Yet, this does not lead to them forgetting how to perform familiar tasks. Experiments on mice have shown that the transformation of the neural areas associated with a learned task continue even after the learning period is ended, with the number of neurons being connected to the task decreasing as time goes by. The storage and energy requirements needed to memorize how to perform the task decrease as a result, but the skill itself is not forgotten. A type of task-related synaptic consolidation occurs and certain synapses are rendered much less plastic than others, leading to long-term knowledge storage.

It is thus reasonable to expect from an AI system that attempts to mimic or even to surpass human-level intelligence not only to reuse old pieces of knowledge to solve new problems, but to do so without forgetting how to solve past ones. The area of machine learning that deals with these model attributes is called **continual** or **life-long learning (CL)** (Ring (1998) [174]. Continual learning differs from transfer learning in three main ways. One is the number of tasks and of the data per task that are usually available. In the case of CL models deal with many more tasks with a much smaller number of samples per task than in transfer learning. Second, the goal of transfer learning is to improve the performance of the model on the predetermined set of target tasks. On the other hand, continual learning's main goal is to structure knowledge acquired from previous tasks in a way that it can easily used by any future task regardless of its characteristics. Finally, in transfer learning settings, the performance of the model on source tasks is not a primary goal of the process, and one may even not be concerned about it at all. However, when performing CL one wants to improve at the downstream tasks but also retain good performance on previously learned tasks.

Unfortunately, NNs' performance on previously learned tasks is known to degrade when they are trained on new tasks, as knowledge about previous tasks that is stored in their weights is corrupted by the new training process. This phenomenon is known as **catastrophic forgetting (CF)** (McCloskey and Cohen (1989) [175], Ratcliff (1990) [176]). To make matters even worse, is is believed that there is a point in the sequential training of NNs on multiple different tasks after which no more knowledge can be stored in their weights (Aljundi et al. (2019) [177]). One must then choose between increasing the model's size to increase its **representational capacity** or accept a decline in the model's performance on either already learned tasks or on new tasks. The later is connected to a famous matter in the field of continual learning, the **stability-plasticity dilemma**, where one has to choose between consolidating previously acquired knowledge and thus minimizing the capacity to acquire new, or providing the model's weights with the freedom to learn new knowledge while accepting the risk of it forgetting what it already knows.

The desired properties of a CL model, as given by Biesialska et al. (2020) [178] are:

- **knowledge retention**: catastrophic forgetting does not occur when learning a new task
- **forward transfer**: the model effectively uses previously acquired knowledge to efficiently

learn a new task

- **backward transfer**: knowledge acquired by a model when learning a new task is effectively used to improve its performance on tasks it has already been trained on
- **on-line learning**: like it happens in real life, data samples should be provided one sample at a time
- **no task boundaries**: humans learn how to perform tasks even when no clear definitions of the tasks have been provided and use a variety of data sources and types to do so; so should the models
- **fixed model capacity**: the storage requirements should remain constant regardless of the number and the difficulty of tasks the model is faced with

Currently, no approach meets all of the aforementioned requirements. Most CL systems focus on achieving knowledge retention, forward transfer and, usually, fixed model capacity. They usually use a set of i.i.d. samples for training whose format is predetermined and doesn't change during the learning process.

4.8.10 Continual Learning Approaches

Rehearsal Methods

Rehearsal methods store the entire or part of the dataset of each previously seen task and periodically retrain the model on them. Memory consolidation in the human brain is known to employ rehearsal methods to stabilize memory acquisition (McClelland et al. (1995) [179]). Yet, this unfortunately increases memory requirements with the number of tasks.

Pseudo-rehearsal methods don't use actual training samples of previous tasks, but store information related to the distribution of past tasks, which are then used to generate new samples.

An slightly different and interesting approach is proposed by Li and Hoiem (2017) [180] that split their model parameters into parameters that are shared by all tasks, $\boldsymbol{\varphi}$, and task-specific ones, $\boldsymbol{\vartheta}_j$, that sit on top of the first. Every time the model is trained on a downstream task T_i, apart from the task-specific parameters that are related to it, $\boldsymbol{\vartheta}_i$, the target tasks' samples are also used to train the task-specific neural structures related to past tasks, $\boldsymbol{\vartheta}_j$, $j = \{1, \cdots, i-1\}$. While $\boldsymbol{\vartheta}_i$ are trained to improve performance on T_i, the other task-specific structures are trained so that they produce the same outputs to the target task's samples before and after the model is trained on T_i, even though the shared parameters $\boldsymbol{\varphi}$ have changed in between. The intuition is that, if their training is successful, then the function that each task-specific structure models does not change much, and CF has thus not occurred.

Regularization Methods

Regularization methods introduce regularization terms that affect the plasticity of the model's parameters helping to prevent CF. Kirkpatrick et al. (2017) [181], for example, attempt to benefit from the **overparameterization** of NNs which means that there are several sets of model weights that lead to a good performance to a target task B. Among them, they assume, there exists at least one $\boldsymbol{\vartheta}_B$, that also ensures good model performance on a source task A. They develop **elastic weight consolidation (EWC)**, that determines the flexibility allowed to each parameter during training by estimating how important this parameter is for previous tasks by using the Fisher information matrix. If a parameter is very important then it is not allowed to

diverge far from its initial position. If it is not, then it is allowed to be freely trained as to assist fitting the dataset of task B.

Such methods implement a notion known as **selective plasticity**. An extreme version of this notion is keeping all previously learned weights frozen to avoid CF.

Instead of using a regularizer one can alternatively adjust the learning rates of parameters to control how fast each parameter is allowed to change.

Knowledge Distillation Methods

Knowledge-distillation methods [182] attempt to reduce memory requirements by transferring the skills of a large model (teacher) to a smaller one (student).

Architectural Methods

Architectural methods are a family of techniques that attempt to prevent CF and achieve forward transfer by employing architectural changes to the models. A common approach is trying to explicitly or implicitly split the model into modular structures, that specialize in a subset of tasks and are not affected when irrelevant tasks are learned. This approach will be discussed in more detail later.

Other works, motivated by the representational capacity argument and the fact that neural structures in the human brain are also explicitly formed to store new knowledge, attempt to dynamically introduce new neural structures to models that have already been trained in order to increase their capacity. Rusu et al. (2016) [183], for example, instantiate a new neural structure, which they refer to as column, whenever a new task is presented, and freeze all previously learned columns to avoid CF. Forward transfer is achieved by the use of lateral connections from the layers of the frozen columns to the layers of the new one. These connections are trained along with the new column. Even though the number of parameters increases quadratically with the number of tasks, they show that the features that new columns learn become increasingly unimportant for the next tasks, as the most relevant features have already been learned by the first columns. Later works attempt to fix this problem and expand the model to new domains [184, 185].

4.8.11 Learning to Continually Learn

Beaulieu et al. (2020) [24] chose a meta-learning approach to tackle the issue of few-shot continual learning. They note that methods that simply rely on heuristics which are hoped to eliminate CF such as, layer and module freezing [183], optimizing Fisher criteria approximations [181], or promoting sparse representations are inconsistent with a fundamental principle of ML, i.e. that one should optimize for what must be achieved, and not trying to accomplish it as a byproduct of another method. Importantly, this is an intuition that significantly contributed to the birth of meta-learning. The objective of their meta-learning algorithm is thus to simultaneously learn new tasks and remember previously seen ones.

Beaulieu et al. (2020) [24] are also inspired by findings of neuroscience that some neural signals play the role of neuromodulators, enabling or disabling synaptic plasticity. This mechanism is important for storing long-term knowledge. They hope that the use of of such a mechanism will prevent CF in a network used to make predictions. This essentially is an application of the conditional computation method (chapter 4.7.5) in transfer settings.

They build on another approach that employs meta-learning to avoid CF, **Online Aware Meta-Learning (OML)** [186]. OML trains a CNN with a MAML-based algorithm and then applies it to sequentially learn different classes of objects, while keeping the convolutional layers frozen to avoid CF. Beaulieu et al. (2020) [24] propose **A Neuromodulated Meta-Learning**

algorithm (ANML) that trains two models, the **prediction network**, parameterized by ϑ_P, responsible for making predictions, and a **neuromodulatory (NM) NN**, parameterized by ϑ_{NM}, that gates the first.

They use the same input for both networks. Each one is a CNN with 3 convolutional layers, followed by a fully-connected one. The final layer of the NM net is of the same size as the fully-connected layer of the prediction net and gates it via element-wise multiplication. This form of gating similar to the way sigmoid functions are used to selectively inhibit signals, chapter 2.6.6. The model is shown in figure 4.21.

They note that their method achieves two things simultaneously, one during backpropagation and another during the forward propagation. First, by inhibiting irrelevant signals they avoid computing gradients for the respective neurons, and thus the respective parameters don't change. This effectively implements a notion called **selective plasticity** and mitigates CF. Moreover, in a CL setting, different parts of the model specialize in different tasks, which may be completely different, such as the tasks of classifying flower specimens vs classifying human sentiment. Having areas with different functionalities activate simultaneously leads to signal interference, leading to drop in model performance on the the new task. By inhibiting signals from irrelevant areas this is avoided. But, if some neural areas trained on past tasks are relevant their signals may be allowed passage enabling forward transfer.

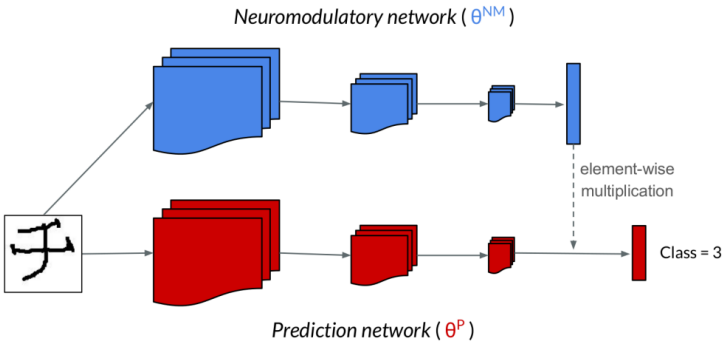


Figure 4.21. *The prediction network is shown in red and the neuromodulatory (NM) net in blue. Both use the image as an input. The final layer of the NM net has the same dimensionality with the fully-connected net of the prediction network, and each of its neurons uses a sigmoid activation function responsible for gating the corresponding unit of the fully-connected layer of the prediction net [24].*

Algorithm

The tasks that are learned are character classes coming from the Omniglot dataset [161], each with its own unique training and test set. One task is equivalent to learning one character class.

To facilitate the following discussion the inner-loop of the meta-training process, during which the model parameters ϑ are trained on a single task's samples, will be referred to as **meta-training training**. The outer loop, during which the meta-parameters φ are trained will be referred to as **meta-training testing**. Similarly meta-testing will be divided into **meta-testing training** and **meta-testing-testing**, during which no updates are performed.

A meta-learning algorithm that is designed to train a model in CL should, in principle, sequentially present to the model in every iteration training data from different tasks and perform meta-training training. After T tasks are shown it should use test samples drawn from these tasks to perform meta-training testing, propagating the gradients back through all of the inner loops. Since this is computationally and storage-wise challenging Javed and White (2019) [186] proposed an alternative which is adopted by Beaulieu et al. (2020) [24]. After the model is trained on the training data of any character class, T_i, adapting its parameters ϑ, it is shown samples from the same class T_i along with test samples from classes T_j, which it has already been trained on $j \in \{1, \cdots, i-1\}$. These are used to update its meta-parameters φ on remembering them. The

set of samples from past tasks is called a **remember set** and it used only for these purpose. The resulting meta-loss is an approximation of the true one.

Beaulieu et al. (2020) [24] trains the model on 20 training examples of each classes during meta-training training performing 20 SGD updates respectively, (chapter 2.4.4), to allow for online learning. During meta-training testing they train the meta-parameters with the same 20 samples of the class that was just learned along with 64 character instances randomly sampled from the remember set.

In the inner loop, the weights pf the prediction network are adjusted to fit the most recent class. The NM net is not trained during meta-training training, but continues to modulate the activations of the neurons of the fully-connected layer while this is trained. On the contrary, all weights are meta-learned during during meta-training testing in a MAML-style (chapter 4.8.6) training process. These weights are used to initialize the model in the next loop. In total, 20,000 outer loops were used to train the model. The process is shown in algorithm 4.4.

ALGORITHM 4.4: *A Neuromodulated Meta-Learning algorithm (ANML)*

Input: $\mathcal{T} \leftarrow$ trajectory of T sequential meta-training tasks
Input: $\boldsymbol{\vartheta}_{NM} \leftarrow$ weights of the NM network
Input: $\boldsymbol{\vartheta}_{P} \leftarrow$ weights of the prediction network
Input: $\alpha, \beta \leftarrow$ learning-rate hyper-parameters
  Initialize $\boldsymbol{\vartheta}_{NM}, \boldsymbol{\vartheta}_{P}$
  for $i = \{1, 2, \cdots\}$ **do** ▷ meta-learning outer-loop
    $S_{traj} = T_i$ ▷ trajectory for inner-loop training
    $S_{rem} \sim \mathcal{T}$ ▷ create remember set
    for $j = \{1, 2, \cdots, k\}$ **do** ▷ meta-learning inner-loop
      $\boldsymbol{\vartheta}_{P}^{(j)} = \boldsymbol{\vartheta}_{P}^{(j-1)} - \beta \nabla_{\boldsymbol{\vartheta}_{P}^{(j-1)}} L(\boldsymbol{\vartheta}_{NM}, \boldsymbol{\vartheta}_{P}^{(j-1)}, S_{traj})$ ▷ SGD on $\boldsymbol{\vartheta}_{P}$
    end for
    $\boldsymbol{\vartheta}_{P,NM} = \boldsymbol{\vartheta}_{P,NM} - \alpha \nabla_{\boldsymbol{\vartheta}_{P,NM}} L(\boldsymbol{\vartheta}_{NM}, \boldsymbol{\vartheta}_{P}^{(k)}, S_{traj}, S_{rem})$ ▷ meta-update on $\boldsymbol{\vartheta}_{P,NM}$ w.r.t.
  final inner-loop model weights $\boldsymbol{\vartheta}_{P}^{(k)}$
  end for

ALGORITHM 4.5: *Meta-Testing Process*

Input: $\mathcal{T} \leftarrow$ trajectory of T sequential meta-testing tasks
Input: $\boldsymbol{\vartheta}_{NM} \leftarrow$ meta-learned weights of the NM net
Input: $\boldsymbol{\vartheta}_{P} \leftarrow$ meta-learned weights of the prediction net
Input: $\beta \leftarrow$ learning-rate hyper-parameter
  $S_{train} = []$
  for $i = \{1, 2, \cdots\}$ **do** ▷ meta-learning outer-loop
    $S_{traj} \sim T_i$ ▷ next meta-testing task
    $S_{train} = S_{train} + S_{traj}$ ▷ add to meta-testing task set
    for $j = \{1, 2, \cdots, k\}$ **do** ▷ meta-learning inner-loop
      $\boldsymbol{\vartheta}_{P} = \boldsymbol{\vartheta}_{P} - \beta \nabla_{\boldsymbol{\vartheta}_{P}} L(\boldsymbol{\vartheta}_{NM}, \boldsymbol{\vartheta}_{P}, S_{traj})$ ▷ SGD on $\boldsymbol{\vartheta}_{P}$
    end for
  end for
  record $L(\boldsymbol{\vartheta}_{NM}, \boldsymbol{\vartheta}_{P}, S_{traj})$ ▷ evaluate final $\boldsymbol{\vartheta}_{P}$ on meta-test train set
  $S_{test} \sim \mathcal{T} - S_{train}$ ▷ get meta-testing test set
  record $L(\boldsymbol{\vartheta}_{NM}, \boldsymbol{\vartheta}_{P}, S_{test})$ ▷ evaluate final $\boldsymbol{\vartheta}_{P}$ on meta-test test set

During meta-testing, the model's convolutional layer and the NM network are kept frozen. The parameters in the final of the prediction model that correspond to the meta-testing classes which are randomly initialized. The model is sequentially trained on Omniglot character classes, i.e. the

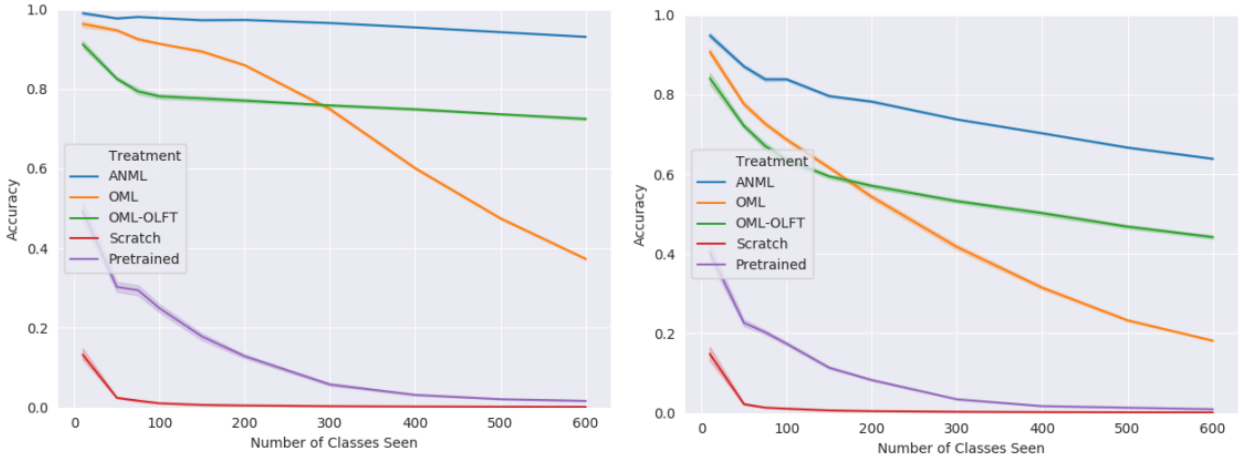
model is trained on 15 training samples of a class it is immediately trained on 15 examples of the new class without re-initializing the recently trained weights. Beaulieu et al. (2020) meta-test the model on up to 600 classes in the aforementioned manner. Therefore, after $15 \times 600 = 9000$ gradient updates the net's weights are $\boldsymbol{\vartheta}_{NM}$ and $\boldsymbol{\vartheta}_{P,9000}$.

The model is evaluated on two-tasks, i.e. on whether it remembers the meta-testing training set of 9000 samples, and on whether it can generalize to an unobserved test set consisting of 5 new examples examples for each of the meta-testing classes. The meta-testing process is shown in algorithm 4.5.

Results

The model's performance is shown in the figure 4.22. The comparing baselines are:

- a randomly initialized net that is then trained on the meta-testing training set data alone, which are presented in an i.i.d. fashion
- a NN pre-trained on the meta-training dataset (includes both meta-training training and meta-training testing image sets shown in an i.i.d. fashion) and then fine-tuned on the meta-testing training set (i.i.d. training)
- OML, that has two fully-connected layers on top, which are fine-tuned during meta-testing while its convolutional layers are kept frozen
- OML-OLFT where only the last of the fully-connected layers is trained during meta-testing
- an interleaved training technique that meta-trains the model the same way as ANML is meta-trained but provides the meta-testing data in a i.i.d. manner, sampling from all meta-testing classes simultaneously. Therefore CF is avoided and the corresponding model is considered to be an upper bound for the ANML method



(a) *Meta-Training* **(b)** *Meta-Testing*

Figure 4.22. ***(a)*** *Accuracy in the task of classification of the meta-testing training images, on which the models have already been trained, for a various number of classes* ***(b)*** *Accuracy in the task of classification of the meta-testing testing images, on which the models have not been trained, for a various number of classes [24].*

ANML clearly outperforms all other techniques both in terms of remembering the training set and in terms of generalizing to unseen instances of its classes. Specifically it only performs 10% worse than the interleaved training technique (not shown in the figures). This difference

is attributed to effect of CF alone and thus the model seems to successfully deal with this issue. Interestingly, the curves corresponding to the ANML and OML-OLFT models have the same slopes along the x-axis, indicating that weight-freezing is significantly contributing in avoiding CF.

ANML and the baselines are only trained once on every sample to preserve the online character of the task. By increasing the number of times the model is trained on every meta-testing training sample and using i.i.d. data for meta-testing training the ANML technique reaches an accuracy level of 75.37%, higher than all other models. Moreover, even though 53% of the neurons of the final layer of the prediction model are on average active before neuromodulation is applied, only 5.9% remain active after the multiplication is performed, as shown in figure 4.23. They contend that this means that sparseness naturally arises without explicitly optimizing for it. Nevertheless, all neurons are used at least once during meta-test training indicating that the available resources are put to good use.

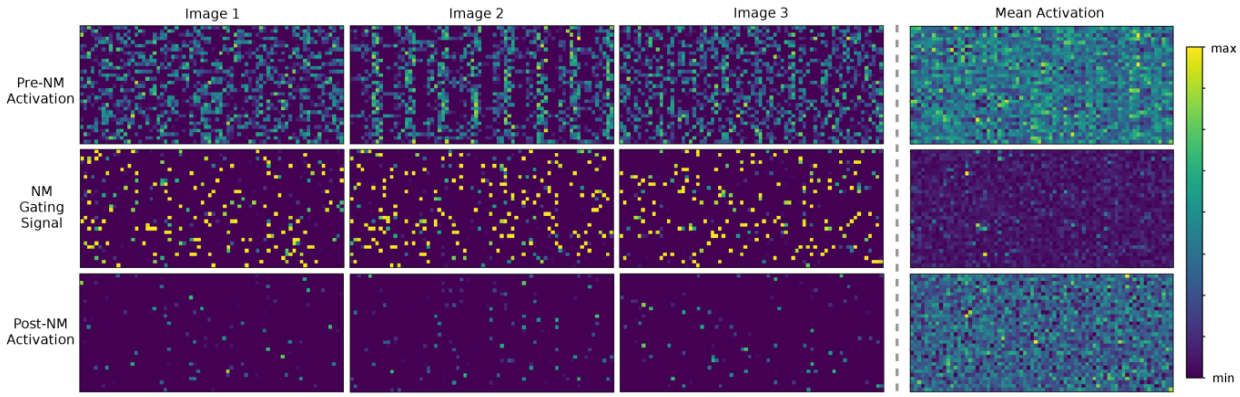


Figure 4.23. *Activations of the final layer of the prediction net shown for three random images of the meta-testing test set before (upper row) and after (bottom row) neuromodulation is applied. The gating signal that is applied in each case is shown in the middle row [24].*

Using the NM net's activations Beaulieu et al. (2020) trained a classifier that scored an accuracy score of 70.9% on the meta-testing test set.

Importantly, as shown in figure 4.24, performance drops as parts of the net that were frozen are trained during meta-test training. Similarly OML-OLFT performs better than OML in the case of many meta-testing classes (figure 4.22). These results are crucial because, if the model is called to learned transfer distributions that are much different from the ones it was trained on, some of the learned features will unavoidably become useless. A larger part of the network will then have to be fine-tuned.

4.9 Inductive Biases for Deep Learning of Higher-Level Cognition

4.9.1 Inspirations From Cognitive Science

Goyal and Bengio (2020) [51] seek to bridge gap between the capabilities of modern neural networks and human intelligence. Their proposals are based on findings of cognitive neuroscience, and especially on research performed on the ability of humans to use, mostly verbalizable, high-level variables and to generalize in few-shot, out-of-distribution settings by combining existing pieces of knowledge.

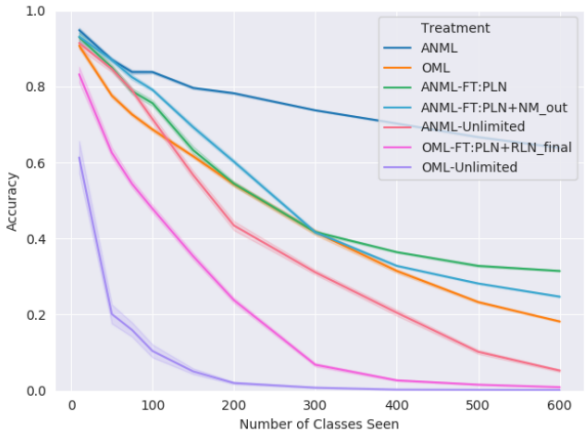


Figure 4.24. *Accuracy on the meta-testing test set for a variety of different versions of the ANML and OML models. Unlimited means that the entire net is trained during meta-testing training. ANML-FT:PLN is the result of training the prediction network, while keeping the parameters of the NM net frozen. ANML-FT:PLN+NM_ out additional trains the final layer of the NM net. OML-FT:PLN+RLN_final trains the 2 fully connects layers of the OML model and the first convolutional one [24].*

A source of inspiration is the mental separation of cognitive processes into System 1 and System 2, which is employed by Kahneman in Thinking Fast and Slow [50], as was discussed in chapter 4.2. They associate System 2 with the conscious manipulation of high-level concepts and System 1 with automatic handling of low-level sensory input. The slow and deliberate functioning of System 2 is believed to be in charge of handling problem distributions that a human might have never observed before, i.e. **out-of-distribution (OOD) generalization**. Pieces of knowledge that are deemed to be relevant to a certain situation are brought to memory as a result of System 1 processes and System 2 is responsible for combining them, possibly in novel ways, while quickly learning pieces of knowledge that are missing.

In addition to that, according to the Global Workspace Theory (GWT [44, 45], chapter 4.5), the brain consists of specialized agents that communicate through a single channel. Each agent can access the channel to transmit information signals to the rest but, only if a criterion judging the information's relevance and importance is met, does the channel broadcast the signals. Baars (1993) [44] supports the idea that after the channel's decision to broadcast the signal humans become conscious of this piece of information and System's 2 processes are applicable. The channel thus plays the role of a bottleneck, i.e. only allows a few agents to simultaneously broadcast signals through the channel. Goyal and Bengio (2020) [51] note that there is a connection between this property of the GWT model and the threshold in the number of concepts a human is able to simultaneously manipulate. This shows the importance of the human ability to create and manipulate high-level concepts to the OOD generalization process, as it enables humans to compress big pieces of information into abstract notions that are represented by only a few variables.

Goyal and Bengio (2020) [51] connect the ability of modern neural nets to automatically (instinctively) recognize and come into conclusions about familiar patterns to the automatic way in which System 1 functions. Therefore, they suggest that machine learning models must improve in performing System's 2 functions in order to achieve human-level intelligence. More specifically, a fundamental capability that neural methods do not yet possess, they note, is the ability to generalize in OOD settings, in cases where no strong supervision is provided, i.e. few samples are given and not all of them may be labeled. They are also unable to create hierarchies of representations, with abstract high-level features, usually related to language, sitting on top.

4.9.2 Current Use of Inductive Biases

Goyal and Bengio (2022) [51] suggest that deep learning has succeeded in part thanks to a set of preferences or priors, which they call **inductive biases**, and have been incorporated into the training processes and the architecture of NNs. This is directly connected to the recurring matter of priors in this thesis (chapter 2.5.2). Convolutional networks have been suggested to implement the prior belief of group equivariance, mainly over space (chapter 2.5.2). The soft-attention mech-

anism has also been explained to implement the notion of equivariance over permutations (chapter 3.7). Finally, weight sharing in RNNs, stems from the assumption of equivariance over time (chapter 2.6). Goyal and Bengio (2020) consider the even more fundamental subjects of distributed representations and deep architectures to implement the beliefs that inputs should be mapped to patterns of features and that complicated functions should result from a composition of simpler ones respectively. All of the above are example of translating preferences into neural architectures.

One can also think of inductive biases as implicit data injections, since they guide the model into preferring some hypotheses over others, essentially playing the role of additional data. This is proven by the fact that biases play an increasingly significant role in scarce data regimes, and become less important when data is abundant. MAP, presented in chapter 2.3.4, is such an example, where the importance of the prior belief over parameters, $p(\boldsymbol{\vartheta})$, decreases as the number of available examples, N, becomes larger. In fact, not all biases are equivalently strong, and some even correspond to exponentially bigger amounts of data injections than others.

Goyal and Bengio (2020) [51] suggest that more biases must be incorporated into deep learning techniques for them to reach human-level intelligence, and that simply increasing model sizes and the quantity of the training data will not suffice.

4.9.3 Requirement for New Training Settings

A way to introduce inductive biases is through the training process. Goyal and Bengio (2022) [51] contend that modern training methods only promote **in-distribution generalization**, i.e. generalization to unseen samples coming from the same distribution with the ones used for training, and that presenting samples in an i.i.d. fashion is not representative of real-world scenarios, where there might exist temporal dependencies between the different training samples.

They thus propose the use of new training and testing settings, where distributions will be non-stationary and statistical dependencies between training data will exist. Under such conditions one is forced to find the relation relation between distributions of tasks used to train the model and also of tasks the model might encounter later in its lifetime, i.e. separate the stationary properties of the training environment that don't vary as new tasks are introduced from the non-stationary ones that do change.

In fact, in order to achieve the above, they promote the use of transfer learning (chapter 4.7.2), meta-learning (chapter 4.8) and continual learning (chapter 4.8.9) settings that study sequential training on more than one different tasks under different scenarios.

4.9.4 Creating A Framework to Study System 2

Goyal and Bengio (2022) [51] adopt ideas from studies on compositionality in order to address the problem of explaining how humans combine reusable pieces of knowledge. **Compositionality** is a notion well known in the field of linguistics (Lake and Baroni (2017) [187], Bahdanau et al. (2018) [188]) as it is an inherent characteristic of language, i.e. the meaning of a new term can be derived from a composition of the meaning of the its comprising terms. Taken to the field of cognitive science, it is related to the process of mentally combining knowledge pieces to address a new task. This process is also called **systematic generalization**, and is enables humans to successfully conduct inference in tasks that could never have come up during training, such as feeling empathetic towards a protagonist of a science-fiction novel the reader has never read before.

In their effort to come up with a mathematical framework that can be used to study the distinct pieces of knowledge individually as well as the various ways in which they can be combined by the human mind to handle novel conditions, Goyal and Bengio (2022) [51] turn to the field of causality (chapter 4.6). They contend that humans maintain a causal model of the world in their

heads. In causal terminology, this assumption means that the pieces of knowledge stored in the human brain can be represented by mechanisms, and the high-level concepts by causal variables. Non-stationarity caused by the appearance of new tasks are modelled by interventions. Note that this is different from the argument that real-world mechanisms can be modeled by SCMs, discussed in chapter 4.6.1. The assumption presented here is that humans model the world using a causal framework in order to successfully perform the various tasks they have to.

4.9.5 Proposing Biases

Taking into consideration the thoughts presented above, e.g. the two thinking modes, GWT, systematic generalization through compositionality and causality etc, Goyal and Bengio (2022) [51] propose a set of inductive biases, some of which are listed below.

High-Level Variables are Verbalizable and can be Linked to Low-Level Ones

The high-level variables that are used in conscious processing are largely verbalizable, or, to be more precise, there is a lossy summary map from these variables to natural language expressions. This assumption effectively constrains the search space for these high-level variables. Variables found lower in the hierarchy as well as the respective neural structures can be represented by modern deep learning models that successfully perform functions similar to System 1's, as discussed in chapter 4.9.1.

To link different levels of the hierarchy, Goyal and Bengio (2022) [51] propose an encoder-decoder-type system (chapter 2.6.5), that may for example read low-level input instances and out higher-level ones. Data for the training of such a system can be obtained by recording the relation between low and high-level variables, as well as its evolution as time progresses.

High-Level Variables are Causal

High-level variables and the relations between them can admit causal factorization, i.e. the variables and the mechanisms connecting them are causal, and changes in these relations are results of interventions.

Based on the work of Daniušis et al. (2010) [157], Peters et al. (2018) [19], these mechanisms are independent and knowledge about them is modular. This means that knowledge about one of them is not informative about any other.

Changes in Distributions are Sparse

Changes in the distribution that result from interventions can be described with a few words. The above, along with the argument that high-level variables are verbalizable, mean that only a few mechanisms need to adapt to account for a change. The parameters of the rest of the mechanisms can be initialized with their old values, as discussed in chapter 4.6.6.

Learning Fast and Slow

Since some properties of the distributions of the various tasks are stationary while others may vary from task to task, the mental system people use to model the world must exhibit similar characteristics. It is true that, as one practices a skill, it progressively migrates from System 2 to System 1, where it consolidates and gains permanent traits.

When trying to build an agent, one should then aspire to have parameters adapting with various learning speeds to the perceived changes. Goyal and Bengio (2022) [51] contend that most weights must be slow to change, enabling the fewer fast weights to be trained fast and efficiently. In

fact, they suggest that models describing relationships between low and high-level variables should change as little as possible, because they mostly correspond to to concepts that do not consist of familiar simpler ones. Humans are known to take longer to learn such concepts. Mechanisms linking high-level causal variables, on the other hand, should be faster to train, since they represent non-stationary attributes of the world that are handled by humans by systematically composing reusable pieces of knowledge.

Representing High-Level Relationships With A Sparse Factor Graph

One then needs an architecture consisting of components that can be easily manipulated and compositionally arranged. Goyal and Bengio (2022) [51] take into consideration the limited capability of the consciousness to broadcast signals simultaneously, according to GWT. They therefore propose to represent the joint distribution of high-level concepts by a sparse factor graph. The causal concepts are represented by nodes, and the mechanisms describing the relationships between them are represented by directed edges, emanating from the nodes representing causes. But, importantly, each the causes used as inputs to a mechanism should not be many.

Use of Generic Rules as Mechanisms

They also propose the use of "generic factors" or "schemas", that interact with causal variables in a way similar to the relation between logic rules and quantifiers, e.g. X may represent a human being and can be bound to a human name such as "Nick". Providing the model with such flexibility allows the organization of knowledge in a way that enables each mechanism to be completely independent of the others, with its own set of parameters. Every time a mechanism is instantiated, the properties of the objects that can be used as its inputs and outputs are defined, and every time it is used, the values of these abstract variables are specified according to the demands of the task at hand.

4.10 A Meta-Transfer Objective for Learning to Disentangle Causal Mechanisms

Bengio et al. (2020) [25] focus on the problem of learning causal models from unknown interventions.

4.10.1 Correct Causal Models are Faster to Adapt

They use the key assumption that changes in distributions are sparse, i.e. interventions resulting to changes in distributions only affect a few of the underlying mechanisms (chapter 4.9.5). They then contend that a correctly trained causal model is faster to adapt to the changes because fewer mechanisms will have to be retrained. In fact, they prove that expected gradient over the transfer distribution of the related log-likelihood w.r.t. to the parameters of the mechanisms/modules that have not changed because of interventions and have been correctly learned during training is zero. Therefore, only the parameters of the modules modelling changed mechanisms will have to be adapted to the transfer distribution, leading to a faster adaptation for the correct causal model.

They show this experimentally by assuming a simple model of two causal variables A and B, whose distributions and relation are unknown. A and B are only known to receive K discrete

values each. The possible factorizations are:

$$P_{A\to B}(A,B) = P_{A\to B}(A)P_{A\to B}(B|A) \tag{4.22a}$$

$$P_{B\to A}(A,B) = P_{B\to A}(B)P_{B\to A}(A|B) \tag{4.22b}$$

each one corresponding to a different causal model (chapter 4.6.5). They set the correct causal model to be $A \to B$.

They use four FFNNs to model the four distributions, with parameters $\boldsymbol{\vartheta}_{A|B}, \boldsymbol{\vartheta}_A, \boldsymbol{\vartheta}_{B|A}, \boldsymbol{\vartheta}_B$, where $\boldsymbol{\vartheta}_{A|B}$ and $\boldsymbol{\vartheta}_{B|A}$ model the K^2 possible value combinations of A and B. As suggested by chapter 4.6.2, there models are indistinguishable when only observational data are available.

After learning the four modules they change the distribution of $P(A)$, keeping $P(B|A)$ fixed. This induces a change in both $P_{B\to A}(B)$ and in $P_{B\to A}(A|B)$, but not in $P_{A\to B}(B|A)$. According to the aforementioned proposition, when the correct causal model $A \to B$ is used, only $O(K)$ parameters of the FFNN modeling $P_{A\to B}(A)$ have to be relearned, whereas, when the anti-causal model $B \to A$ is used the computational complexity of the training is increased to $O(K^2)$, which essentially corresponds to the entire model being retrained. This is shown in figure 4.25, where the causal model requires much fewer samples to adapt to the change.

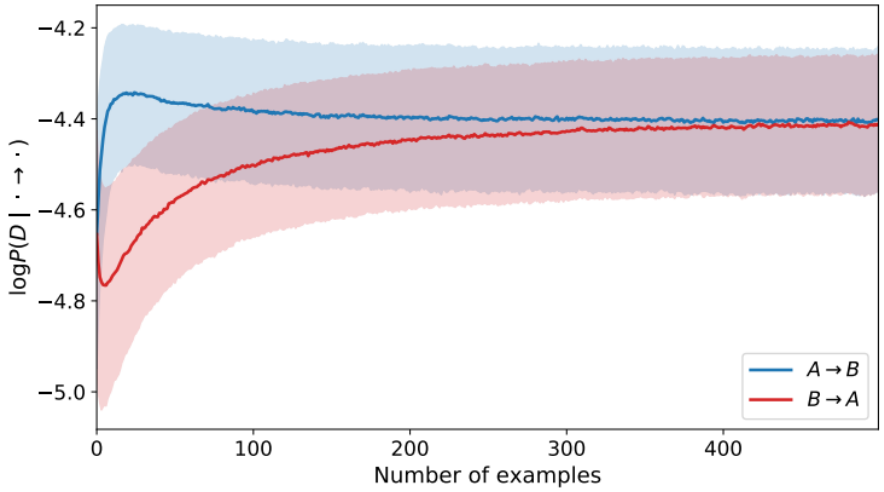


Figure 4.25. *Log-likelihood of the transfer data computing using the causal $A \to B$ and the anti-causal $B \to A$ models w.r.t. the number of examples from the transfer distribution shown to them. The causal models is shown to adapt much faster than the anti-causal one [25].*

4.10.2 Using Adaptation Speed as a Meta-Learning Objective

They then attempt to use this difference in adaptation speeds to learn the causal graph from interventional data. More specifically, they propose using adaptation speed as a meta-learning signal to train meta-parameters, with each meta-parameter corresponding to an edge between two causal variables. Each meta-parameter expressed the belief about the direction of the respective edge, i.e. indicating which variable is a cause and which is the effect.

They experiment with the two variable case, they define a structural meta-parameter γ and set $P(A \to B) = \sigma(\gamma)$. Therefore $P(B \to A) = 1 - \sigma(\gamma)$. They then define the meta-loss (regret) as:

$$R = -\log(\sigma(\gamma)L_{A\to B} + (1-\sigma(\gamma))L_{B\to A}) \tag{4.23}$$

where $L_{A\to B}, L_{B\to A}$ the online likelihoods of both models respectively on the transfer data D_2:

$$L_{A\to B} = \prod_{t=1}^{N_{D_2}} P_{A\to B}(a_t, b_t; \theta_t) \tag{4.24a}$$

$$L_{B\to A} = \prod_{t=1}^{N_{D_2}} P_{B\to A}(a_t, b_t; \theta_t) \tag{4.24b}$$

and $\{(a_t, b_t)\}_{N_{D_2}}$ the set of transfer examples for a given training episode. θ_t represents the model's parameters at t and $P_{model}(a, b; \theta)$ is the likelihood of the sample (a, b) under the respective model.

The meta-training signal is $\partial R \setminus \partial\gamma$ and it is used to update γ with an SGD update every time an episode (inner-loop) of N_{D_2} transfer examples is completed. The intuition is that during the inner loop the correct model will have adapted to the transfer data better than the incorrect one. As a result the corresponding loss function L_{model} will be larger and γ will take a step towards the respective direction during the outer loop. They prove that SGD indeed converges to the true causal model, i.e. $\sigma(\gamma) = 1$ if $E_{D_2}[\log_{A\to B}] > E_{D_2}[\log_{B\to A}]$ and $\sigma(\gamma) = 0$ otherwise. They also prove it experimentally as shown in figure 4.26. The same is shown for continuous multi-modal variables.

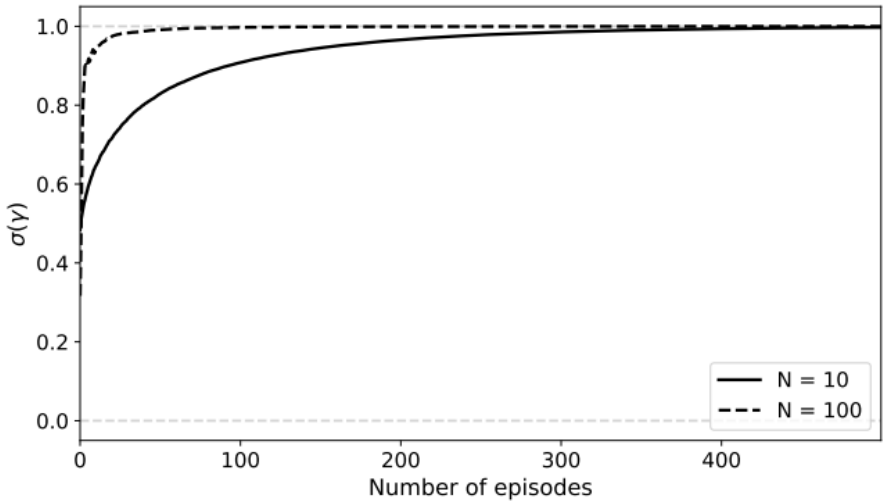


Figure 4.26. *The evolution of $\sigma(\gamma)$ during meta-learning, where $A \to B$ is the correct causal model [25].*

Interestingly, the smaller the adaptation task in terms of training data, the stronger meta-learning signal since, given enough data, both models converge to an acceptable solution, and thus distinguishing them becomes difficult. This advocates the use of meta-learning as a training framework to discover causal structures.

4.10.3 Disentangling Causal Variables

Bengio et al. (2020) [25] then turn to the important problem of discovering causal variables from sensory input. Their assumption regarding a sparseness of interventions is only applicable in causal structures but, usually, data is provided in the form of low-level sensory input. The problem of discovering causal variables is closely connected to the one of disentangling factors of variation (Bengio et al. (2012) [189], Mathieu et al. (2016) [190]).

They consider the simple case of an input consisting of two variables, X and Y, which are related to two causal variables A and B via a decoder. The decoder is known to implement a

rotation with parameter\angle of rotation θ_D, whose value is unknown:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = R(\theta_D) \begin{bmatrix} A \\ B \end{bmatrix} \tag{4.25}$$

They use an encoder to translate from the input space of "low-level variables" to the space of "higher-level" ones:

$$\begin{bmatrix} U \\ V \end{bmatrix} = R(\theta_D) \begin{bmatrix} X \\ Y \end{bmatrix} \tag{4.26}$$

They treat θ_E as an additional meta-parameter and meta-learn it alongside γ, i.e. during the outer-loop both γ and θ_E are updates with SGD. They verify experimentally that the correct causal variables and structure are learned (figure 4.27).

The intuition here is that the correct causal model ensures the fastest adaptation. Thus, by optimizing for adaptation speed the causal model naturally comes up.

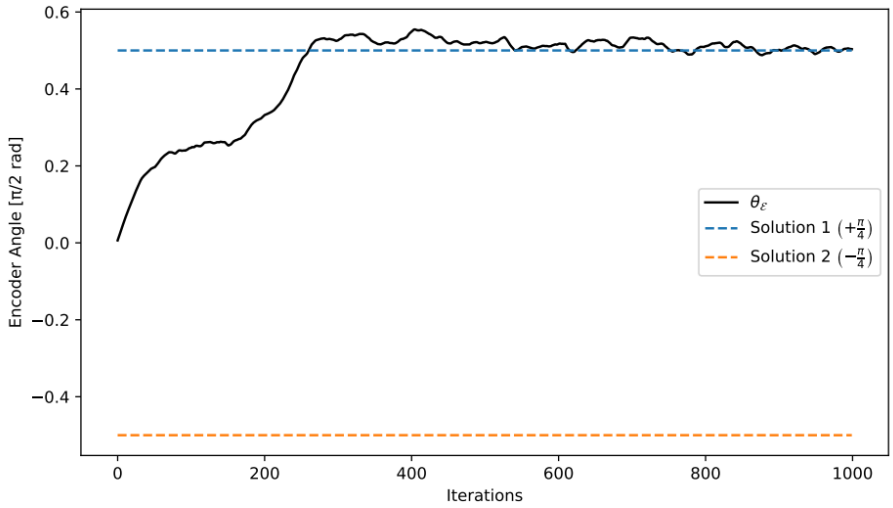


Figure 4.27. *Both the blue and the orange line indicate valid solutions. After a short period of adaptation the first solution is found [25].*

4.11 Recurrent Independent Mechanisms

Goyal et al. (2019) [2], inspired by the notion of independent causal mechanisms describing relations between real-world causal variables, extend the independence assumption to dynamic systems, which they believe to consist from simpler and independent dynamic processes. Even though they are independent, these processes are assumed to sparsely interact with each other. Goyal et al. (2019) [2] view these interactions as the source of the complexity of dynamic systems.

Humans are only able to focus on a limited number of tasks simultaneously. Goyal et al. (2019) [2] therefore make the additional assumption that, at a given moment, humans monitor a subset of the ongoing dynamic processes, which seem to be most interesting at that particular moment. Therefore, an agent trying to mimic a human must exhibit similar behaviour. They contend that implementing this bias will result to faster adaptation to non-stationarities, as only actively observed processes will have their parameters updated. This is similar to the argument presented in Bengio et al. (2020) [25] (chapter 4.10.1). They also suggest that it will reduce interference among processes during forward propagation, as only active and thus informative processes will be able

to communicate their pieces of knowledge along the "bottleneck". A similar idea was presented by Beaulieu et al. (2020) [24] (chapter 4.8.11). They test their model in OOD generalization using various tasks to prove the validity of their assumptions and the effectiveness of their model.

4.11.1 The Model

Goyal et al. (2019) [2] propose **Recurrent Independent Mechanisms (RIMs)**, a recurrent model that implements the inductive biases of independent processes, sparse activations and sparse interactions in an attempt to discover these processes during training and achieve faster and better adaptation in OOD settings. A RIMs model consist of K recurrent modules, each one with its own parameters to allow for module independence (figure 4.28).

Selective Activation with Competition

In order to implement the inductive bias of sparse activations, they try to establish a form of competition among the RIMs, similarly to the work of Parascandolo et al. (2017) [20] (chapter 4.6.7). According to the **Biased Competition Theory** (Desimone and Duncan (1995) [191]), brain systems related to vision compete for brain resources. Basal ganglia (chapter 4.3.2) have been also found to be responsible for choosing which neural signals to inhibit and which to grant access to the next computational stage. Parascandolo et al. (2017) [20] notes that competition leads to specialization of the competing modules, which is in accordance with the effort of Goyal et al. (2019) [2] to discover individual dynamic processes through a competition process.

Nevertheless, in contrast to Parascandolo et al. (2017) [20], Goyal et al. (2019) [2] use attention mechanisms to create competition among the RIMs. The input at time step t, $\mathbf{I}_t$, is considered to be a matrix consisting of row vectors representing a set of objects. A row vector of zeros is concatenated to the matrix creating $\mathbf{X}_t = \emptyset \oplus \mathbf{I}_t$. For each one of the objects a key vector, $\mathbf{K} = \mathbf{X}\mathbf{W}^e$, encoding it properties, and a value vector, $\mathbf{V} = \mathbf{X}\mathbf{W}^v$, encoding the contained information, are computed. For each RIM k, a query vector (or many, in the case of multi-head attention), $\mathbf{Q}_k = \mathbf{h}_{t,k}\mathbf{W}^q_k$, is generated, where $\mathbf{h}_{t,k}$ is its hidden state at time t. Soft-attention is used to compute the input for each RIM:

$$A_{in,k} = \text{softmax}(\frac{\mathbf{h}_{t,k}\mathbf{W}^q_k(\mathbf{X}\mathbf{W}^e)^t}{\sqrt{D_{key}}})\mathbf{X}\mathbf{W}^v,\ k \in \{1, \cdots, k_T\} \tag{4.27}$$

Each RIM k is thus given the freedom to select its own input, depending on its interests at the time, as these are defined by $\mathbf{h}_{t,k}$ and $\mathbf{W}^q_k$. Competition is then established by deciding to activate only k_A of the k_T available RIMs, based on the attention weight each one's input mechanism has assigned to the zero vector, $\emptyset$. The intuition behind this metric is that RIMs that are most interested to the input are hypothesized to assign a larger attention weight to its elements and thus a smaller one on $\emptyset$, and these RIMs are the ones one must activate. In the case of multi-head attention, the average of the attention weights assigned to $\emptyset$ by each attention head of the input mechanism of a certain RIM is used to rank the corresponding RIM. The set of active RIMs is denoted as S_t.

Goyal et al. (2019) [2] also employ competition among the RIMs when the various spatial positions of a spatially structured input. Again, k_s out of k_T RIMs are activated for each spatial position depending on the attention each RIM has paid to it.

The hidden states of the inactive RIMs are not updated:

$$\mathbf{h}_{t+1,k} = \mathbf{h}_{t,k},\ \forall k \notin S_t \tag{4.28}$$

Yet gradients do flow through them. The hidden states of the active RIMs are updated according to the equations of the respective recurrent model:

$$\tilde{\mathbf{h}}_{t,k} = D_k(\mathbf{h}_{t,k}, A_{in,k}; \vartheta_k), \forall k \in S_t \tag{4.29}$$

Sparse Interactions

The active RIMs are also able to interact with other RIMs, by reading information from their hidden states. This inductive bias is implemented by another attention mechanism:

$$\mathbf{h}_{t+1,k} = \text{softmax}(\frac{\mathbf{Q}_{t,k}\mathbf{K}_{t,:}^T}{\sqrt{D_{key}}})\mathbf{V}_{t,:} + \tilde{\mathbf{h}}_{t,k}, \forall k \in S_t \tag{4.30}$$

where a residual connection is used to assuage the vanishing gradients problem. The queries, keys and values for each RIM are computed using its hidden state along with weight matrices defined for this purpose:

$$\mathbf{Q}_{t,k} = \tilde{\mathbf{W}}_k^q \tilde{\mathbf{h}}_{t,k}, \forall k \in S_t \tag{4.31a}$$
$$\mathbf{K}_{t,k} = \tilde{\mathbf{W}}_k^e \tilde{\mathbf{h}}_{t,k}, \forall k \tag{4.31b}$$
$$\mathbf{V}_{t,k} = \tilde{\mathbf{W}}_k^v \tilde{\mathbf{h}}_{t,k}, \forall k \tag{4.31c}$$

Sparsity of interactions is imposed by allowing each RIM k, $k \in S_t$, to consider information coming from a subset of RIMs to which most attention has been paid by k.

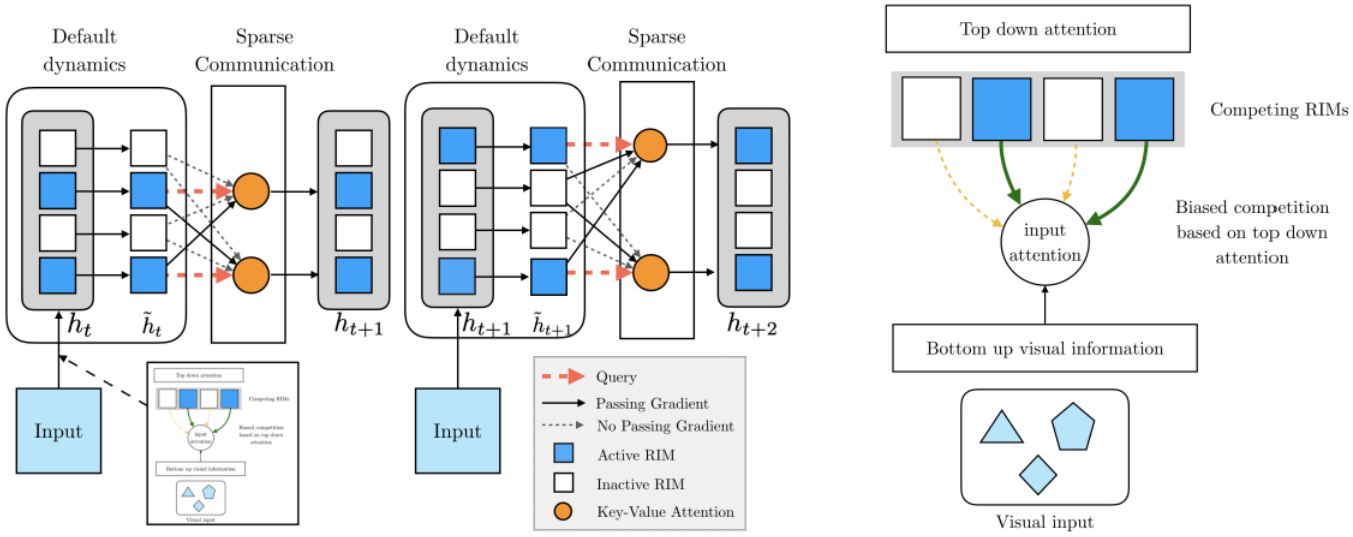


Figure 4.28. *Overview of a model with 4 RIMs. The mechanisms attend to the input elements and the ones that pay most attention to them are activated (**right**). The active mechanisms (with blue) follow their default dynamics and sparsely (continuous vs dashed lines) interact with the other ones (**left**) [2].*

4.11.2 Experiments

Goyal et al. (2019) conduct several experiments to test whether RIMs are able to generalize successfully under OOD conditions.

Temporal Patterns

They first test their ability to generalize over varying temporal patterns. They employ a **copying task** that involves showing the model a short sequence of characters, followed by a 50 consecutive blank inputs. The model is then asked to output the sequence it read in the beginning. A RIMs model with 6 RIMs and 100 neural units per RIM is compared to an LSTM model with 600 units. The results are summarized in table 4.1. While all models succeed in in-distribution generalization, only RIMs are able to generalize to the case in which the number of blank inputs is increased to 200 time steps.

Copying	k_T	k_A	h_{size}	**Train(50)** CE	**Test(200)** CE
RIMs	6	4	600	**0.00**	**0.00**
	6	3	600	**0.00**	**0.00**
	6	2	600	**0.00**	**0.00**
	5	2	500	**0.00**	**0.00**
LSTMs	-	-	300	**0.00**	4.32
	-	-	600	**0.00**	3.56
NTM	-	-	-	0.00	2.54
RMC	-	-	-	0.00	0.13
Transformers	-	-	-	0.00	0.54

Table 4.1. *Cross-entropy loss of various versions of the RIMs model on training and test sets of the copying task. The performance of two LSTMs, a neural Turing machine [34], a relational memory core [26] and a transformer model are also recorded.*

They also train the same models on a **sequential MNIST resolution task**, where the goal is to classify MNIST digits that are provided as sequences of pixels to the model. The models are then asked to generalize to images of different resolutions ($16 \times 16, 19 \times 19, 24 \times 24$) than those observed during training (14×14). It is expected from the RIMs model to have a subset of RIMs specializing in pixels that are not part of the digit and a different subset specializing in pixels that are part of them. RIMs are shown in generalize better to images of higher resolutions (table 4.1) as they keep RIMs that store pixel-related information inactive while pixels that not contain digits are being processed.

Sequential MNIST	k_T	k_A	h_{size}	**16 × 16** Accuracy	**19 × 19** Accuracy	**24 × 24** Accuracy
RIMs	6	6	600	85.5	56.2	30.9
	6	5	600	88.3	43.1	22.1
	6	4	600	**90.0**	**73.4**	**38.1**
LSTMs	-	-	300	86.8	42.3	25.2
	-	-	600	84.5	52.2	21.9
EntNet	-	-	-	89.2	52.4	23.5
RMC	-	-	-	89.58	54.23	27.75
DNC	-	-	-	87.2	44.1	19.8
Transformers	-	-	-	**91.2**	51.6	22.9

Table 4.1. *Cross-entropy loss of various versions of the RIMs model on training and test sets of the sequential MNIST task. The performance of two LSTMs, a recurrent entity network [35], a relational memory core [26], a NN with a dynamic external memory [36] and a transformer model are also recorded.*

The ablation studies reveal the important contribution of the selective activation and sparse interaction biases to the OOD generalization abilities of RIMs.

Bouncing Balls

The model is also tested OOD generalization in the object detection task called **Bouncing Balls Experiment (BBE)**. BBE simulates the movement and interaction of balls of various sizes

and weights that follow Newtonian physics.

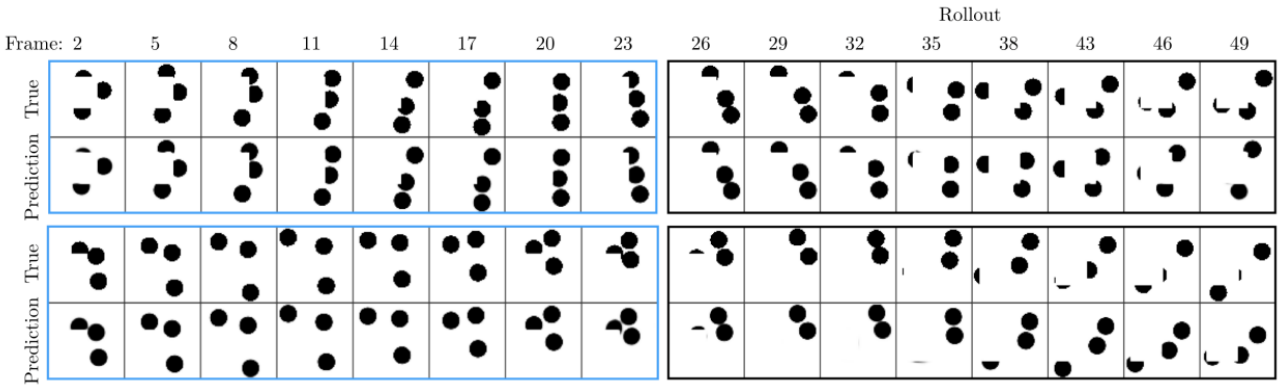


Figure 4.29. *Predictions for two different trajectories (up and down) of three balls in an environment where a curtain is applied. On the left the side of the figure the predictions of the model when the true frames are used as input are shown. On the right side (rollout) the model uses its previous output as input [2].*

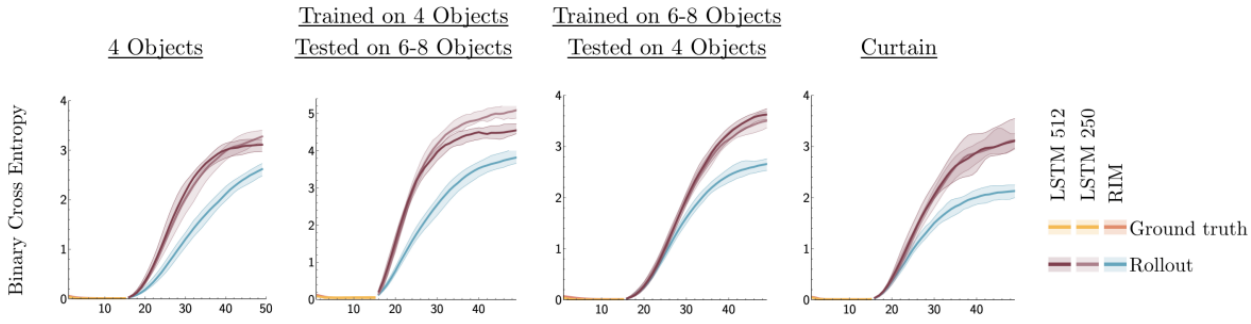


Figure 4.30. *The first 15 frames of the actual ball movements are used as input to the models. Then the models enter an auto-regressive mode, using their past outputs as inputs (rollout). The RIMs model's cross entropy error on the balls' positions is much lower than the LSTMs' one, indicating the model's ability to generalize both under in-distribution and OOD settings [2].*

The models are trained in a teacher-forcing manner (chapter 2.6.2) to predict the next ball position for each ball, and are then tested on OOD generalization on ball environments with different numbers of interacting balls than those that were used during training (6-8 vs 4, 4 vs 6-8) and on predicting the movement of balls in cases where part of the input image is not available to the model (curtain), e.g. the model may not be visible for a while and appear again after a number of steps, as depicted in figure 4.29. As shown in figures 4.30 and 4.31 the model generalizes much better than the other baselines under such circumstances.

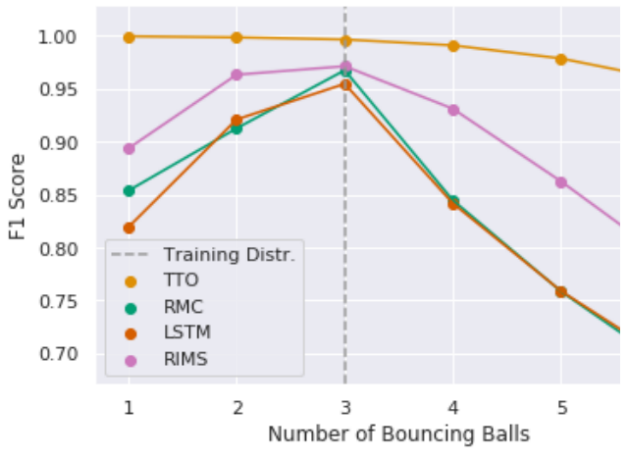


Figure 4.31. *OOD generalization capability as indicated by the F1 score, compared to LSTM and RMC [26] on another partial observation video prediction task. All models were trained on a three-ball setting. TTO is the time travelling oracle that has access to the real dynamics and does not simulate them like the other models (upper bound) [2].*

Reinforcement Learning Problems: Atari

Goyal et al. (2019) [2] test RIMs on RL problems, which they consider a natural fit to RIMs, since, as the agent learns it naturally finds itself in states it has never seen before. They train an RL agent implemented by a model consisting of 6 RIMs ($k_A = 4 - 5$) on Atari games and record its scores. They report improved results over an LSTM baseline (figure 4.32). They also attempt to pre-train their model on 3 source tasks and transfer learn the knowledge to 12 random tasks, hitting a record of 9/12 positive transfer examples using RIMs. The use of LSTMs on the other hand only leads to positive transfer in 3 out of the 12 games.

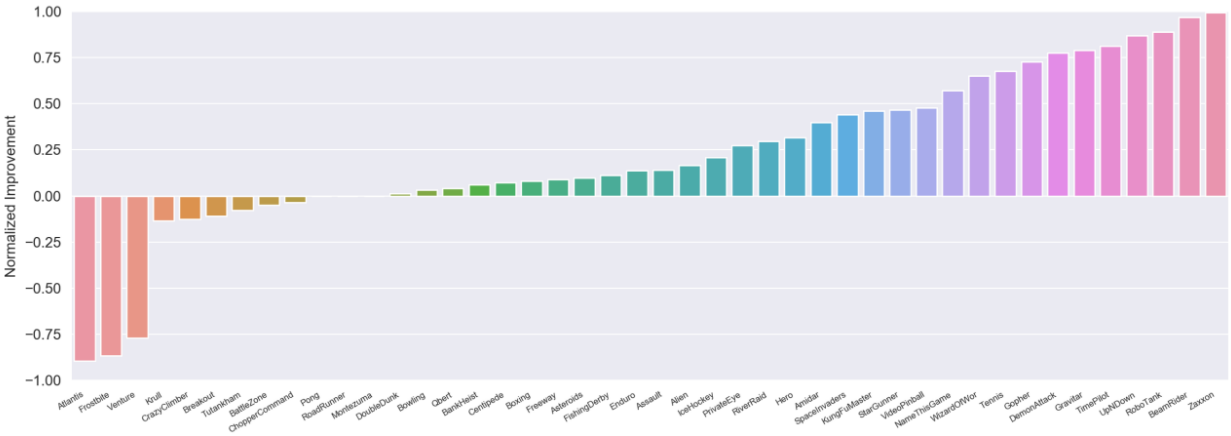


Figure 4.32. *Relative score improvement over an LSTM based model across all Atari games averaged over 3 trials per game [2].*

4.12 Neural Interpreters

Rahaman et al. (2021) [27] focus on the problem of enabling NNs to reuse modules wherever and whenever they deem necessary. They note that modern NNs lack flexibility when it comes to reusing knowledge in different computational stages. For example, convolutional kernels in CNNs and weights of RNNs are only reused laterally and vertically respectively. They thus propose a new architecture, **Neural Interpreters (NIs)**, which, they contend, can flexibly reuse and combine individual pieces of knowledge. Both the computational units and the neural structures responsible for routing information through the computational units are learned from standard supervised learning.

4.12.1 The Model

Neural Interpreters accept as inputs sets of vectors $\{\mathbf{x}_i\}_i$, $\mathbf{x}_i \in \mathbb{R}^{D_{in}}$, and output sets of vectors $\{\mathbf{y}_i\}_i$, $\mathbf{y}_i \in \mathbb{R}^{D_{out}}$ of the same cardinality, just like BERT (chapter 3.9.3). Their composing layers are called **scripts** (figure 4.33):

$$\{\mathbf{y}_j\}_j = \text{NI}(\{\mathbf{x}_i\}_i) = [\text{Script}_{L_s} \circ \cdots \circ (L_s \text{ times}) \circ \cdots \circ \text{Script}_1](\{\mathbf{x}_i\}_i) \tag{4.32}$$

The parameters of two different scripts are distinct. Next, the composing parts of each script will be discussed:

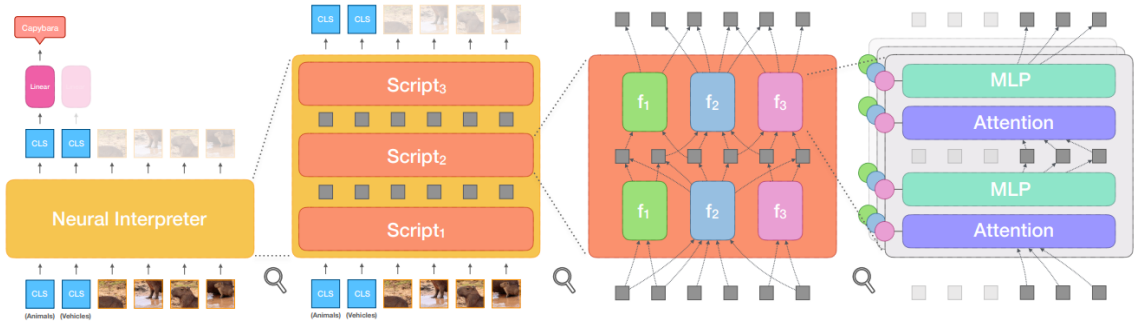


Figure 4.33. *The overview of the a neural interpreter along with its inputs and outputs are shown in the leftmost figure. The CLS tokens are indicated in blue and a linear classification head is attached on top of the output corresponding to each one. In the center left the constituting scripts are shown. Each one of the three scripts uses a separate set of parameters. In the center-right the inner workings of a script are shown. It entails two function iterations and three functions. All parameters are shared between function iterations, but rerouting is performed before each one. In the the rightmost figure the consisting LOCs are shown, conditioned on the third function's code vector (pink). The two other computational streams, for the green and blow function, run in parallel with the first one and are shown on the back. The routing of the input elements is common for all LOCs of the same function iteration that are conditioned on the same vector. The three rightmost input elements are the only ones that are allowed access to the pink function as shown in the figure. Residual connections do exists but are not shown [27].*

Functions

Functions are the computational units of each script, consisting of two vectors, $\mathbf{s}_u$ and $\mathbf{c}_u$, i.e. $f_u = (\mathbf{s}_u, \mathbf{c}_u)$, $u \in \{1, \cdots, N_f\}$. $\mathbf{s}_u$ is the **signature** of the function, meaning that it encodes its properties and acts as an interface. $\mathbf{c}_u$ is the **code vector**, it encodes its instructions regarding the computations that it performs.

Type Matching and Inference

The **type matching** mechanism controls the routing of each input vector through the functions of the corresponding script. It first uses an FFNN to map an input vector $\mathbf{x}_i$ to another vector $\mathbf{t}_i$, $\mathbf{t}_i = \text{TypeMatching}(\mathbf{x}_i)$, which encodes its properties \ type. Using the cosine similarity between $\mathbf{t}_i$ and each of the functions' signatures $\mathbf{s}_u$, $u \in \{1, \cdots, N_f\}$, it determines the compatibility between $\mathbf{x}_i$ and each one of the functions:

$$d_T(\mathbf{s}_u, \mathbf{t}_i) = 1 - \mathbf{s}_u \mathbf{t}_i,\ \mathbf{s}_u, \mathbf{t}_i \in \mathcal{T} \tag{4.33}$$

A hyper-parameter τ, which is called a **truncation parameter**, is compared to d_T and allows access to f_u only to elements for which $d_T(\mathbf{s}_u, \mathbf{t}_i) < \tau$. A compatibility matrix is computed for the ones that are granted access to f_u as follows:

$$C_{ui} = \frac{\tilde{C}_{ui}}{\epsilon + \sum_u \tilde{C}_{ui}},\ \tilde{C}_{ui} = exp[-\frac{d_T(\mathbf{s}_u, \mathbf{t}_i)}{\sigma}],\ \text{if } d_T(\mathbf{s}_u, \mathbf{t}_i), \text{ else } 0 \tag{4.34}$$

where ϵ is a small positive number that is added for numerical stability purposes, and σ is a learnable parameter. C_{ui} is used in the routing process performed by the soft-attention mechanism that will be discussed later.

ModLin and ModMLP

Rahaman et al. (2021) [27] refer to the **modulated linear layers (ModLin)** as programmable neural modules. Essentially the consist of 3 learnable weight matrices: $\mathbf{W} \in \mathbb{R}^{D_{out} \times D_{in}}$, $\mathbf{b} \in \mathbb{R}^{D_{out}}$ and $\mathbf{W}_c \in \mathbb{R}^{D_{in} \times D_{cond}}$ that are shared among functions of the same layer. Each function f_u can program a ModLin via its code vector:

$$\mathbf{y}_{ui} = \text{ModLin}(\mathbf{x}_i, \mathbf{c}_u) = \mathbf{W}(\mathbf{x}_i \otimes \text{LayerNorm}(\mathbf{W}_c \mathbf{c}_u)) + \mathbf{b} \tag{4.35}$$

where $\otimes$ symbolizes the element-wise product.

Modulated MLPs (ModMLPs) are then defined as:

$$\mathbf{y}_{ui} = \text{ModMLP}(\mathbf{x}_i, \mathbf{c}_u) = [\text{ModLin}_{L_M}(\cdot, \mathbf{c}_u) \circ \text{Activation} \circ \cdots \circ \text{Activation} \circ \text{ModLin}_1(\cdot, \mathbf{c}_u)](\mathbf{x}_i) \tag{4.36}$$

where L_M ModLin layers that use the same code vectors are stacked one on top of the other with activation functions between them.

ModAttn

Modulated attention (ModAttn) layers perform self-attention on the each elements of each function separately. The code vector of the corresponding function, along with ModLin layers that are different for every head, are used to compute query, key and value vectors:

$$\mathbf{q}_{uhi} = \text{ModLin}^h_{query}(\mathbf{x}_i, \mathbf{c}_u) \tag{4.37a}$$

$$\mathbf{k}_{uhi} = \text{ModLin}^h_{key}(\mathbf{x}_i, \mathbf{c}_u) \tag{4.37b}$$

$$\mathbf{v}_{uhi} = \text{ModLin}^h_{value}(\mathbf{x}_i, \mathbf{c}_u) \tag{4.37c}$$

Self-attention weights for each pair of elements $\mathbf{x}_i, \mathbf{x}_j$ that have been accepted by a function f_u are then computed at head h:

$$\mathbf{W}_{uhij} = \frac{\tilde{\mathbf{W}}_{uhij}}{\epsilon + \tilde{\mathbf{W}}_{uhij}}, \; \tilde{\mathbf{W}}_{uhij} = C_{ui} C_{uj} \text{softmax}_j\left(\frac{\mathbf{q}_{uhi} \mathbf{k}_{uhj}}{\sqrt{D_{key}}}\right) \tag{4.38}$$

where softmax_j normalizes across j. Notice that weights are also multiplied by the compatibility values between the elements under consideration and the corresponding function.

The computation of the attention values then proceeds as usual with the computation of the appropriate outputs:

$$\tilde{\mathbf{y}}_{uhi} = \sum_j \mathbf{W}_{uhij} \mathbf{v}_{uhj} \tag{4.39}$$

and ends with a linear projection to the concatenated outputs of the various heads per function and per element:

$$\mathbf{y}_{ui} = \text{ModLin}(\tilde{\mathbf{y}}_{u,1:H}; \mathbf{c}_u), \text{ where } H \text{ the total number of heads} \tag{4.40}$$

Lines of Code

A **Line of code (LOC)** consists of a ModAttn layer followed by a ModMLP layer, both conditioned on the same code vector and with residual layers around each one, similarly to the

transformer architecture:

$$\tilde{\mathbf{a}}_{ui} = \text{ModAttn}(\{\text{LayerNorm}(\mathbf{x}_{uj})\}_j; \mathbf{c}_u, \{C_{uj}\}_j) \tag{4.41a}$$

$$\mathbf{a}_{ui} = \mathbf{x}_{ui} + C_{ui}\tilde{\mathbf{a}}_{ui} \tag{4.41b}$$

$$\tilde{\mathbf{b}}_{ui} = \text{ModMLP}(\text{LayerNorm}(\mathbf{a}_{ui}; \mathbf{c}_u) \tag{4.41c}$$

$$\mathbf{y}_{ui} = \mathbf{a}_{ui} + C_{ui}\tilde{\mathbf{b}}_{ui} \tag{4.41d}$$

which means that $\mathbf{y}_{ui} = \mathbf{x}_{ui}$ if $C_{ui} = 0$, so an input vector is not affected by a function it has not been granted access to. Each LOC corresponds to a separate transformer-like computational stream, modulated by a specific function code.

Interpreter

An **interpreter** layer is a stack of L_L LOCs, which share the same functions and compatibility matrices. The output of the final LOC consists of N_f copies of each input element, each one processed by one of the N_f functions. The final representation of each input vector is the weighted sum of these processed copies, using the corresponding compatibility values as weights:

$$\mathbf{y}_i = \mathbf{x}_i + \sum_u C_{ui}(\text{LOC}_{L_L} \circ \cdots \circ (L_L \text{ times}) \circ \cdots \circ \text{LOC}_1)(\{\mathbf{x}_j\}_j; \mathbf{c}_u, \{C_{uj}\}_j) \tag{4.42}$$

The use of the same computational module, i.e. code vectors, by multiple LOCs implements the notion of knowledge reuse addressed in the introduction.

Function Iteration

To allow for a reuse of the computational modules one can stack copies of an interpreter and the type matching mechanism on top of one another. A function iteration is defined as follows:

$$\{\mathbf{y}_i\}_i = \text{FnIter}(\mathbf{x}_j\}_j) = \text{Interpreter}(\mathbf{x}_j\}_j; \mathbf{c}_u, \{C_{uj}\}_{u,j}),\ C_{ui} = \text{TypeMatching}(\mathbf{s}_u, \mathbf{x}_i) \tag{4.43}$$

Multiple function iterations constitute a script, which is, as stated in the beginning, the building block of the NI model:

$$\{\mathbf{y}_i\}_i = (\text{FnIter} \circ \cdots \circ (L_I \text{ times}) \circ \cdots \circ \text{FnIter})(\{\mathbf{x}_j\}_j) \tag{4.44}$$

4.12.2 Module Collapse

A common problem that deep learning algorithms trying to organize neural architectures into modular structures is **module collapse**. Module collapse may occur when the model ends some using only a few of the existing modules and thus fails to benefit from the available expressive power (Kirsch et al. (2018) [192]). Similarly, Parascandolo et al. (2017) [20] noticed that during some runs, a subset of the mechanisms did not specialize to any transformation and remained idle.

Another type of module collapse involves multiple modules converging to a single prototype, and essentially start functioning in the same manner. Rahaman et al. (2021) [27] seek to prevent this form of module collapse from occurring by freezing the function signatures at initialization to keep them separate. They believe that this may force distinct modules into learning different input types and thus developing different specializations.

4.12.3 Experiments

They train their model in three different tasks, chosen to evaluate various model aspects.

A Regression Task with Known Primitives

In order to test whether the learned functions are reusable they train a NI model on a task whose data are by definition created from compositions of known primitives. The task is the regression problem of learning **Fuzzy Boolean Expressions**. They use 5 variables and 3 primitives whose definitions follow:

$$\text{and}(x_i, x_j) = x_i x_j \tag{4.45a}$$

$$\text{not}(x_i) = \overline{x}_i = -x_i \tag{4.45b}$$

$$\text{or}(x_i, x_j) = \overline{\overline{x}_i \overline{x}_j} \tag{4.45c}$$

and consider 30 random Boolean functions $\{f_i\}_{i=1}^{30}$, $f_i : [0,1]^5 \to [0,1]$. 20 of them are learned during a pre-training phase and 10 during an adaptation phase.

During pre-training, they use input sets of 25 vectors, 5 for the 5 variables and 20 learnable CLS vectors, each one corresponding to a training function. A sample consists of a combination of values for the 5 variables and of the 20 outputs of the training functions when these values are used as inputs. A regression head that is shared among functions uses as input the output of the NI corresponding to each of the CLS tokens and outputs a prediction.

To test the learning of reusable modules, they fine-tune 3 versions of the model:

i) freezing all weights and only fine-tune the 10 CLS tokens corresponding to 10 new functions

ii) freezing all weights except the 10 CLS tokens and the parameters of the type matching mechanism (function signatures and them parameters of the type inference mechanism)

iii) fine-tuning the entire model

The idea is that if the function codes have encoded useful and recomposable primitives during the pre-training phase then a simple rewiring during fine-tuning, that is performed in the second case (ii), will suffice to learn the new functions. This exactly is shown in table 4.5, where the second model scores much higher than the first one and only slightly worse than the third one.

Parameter Set	R^2
All Parameters (Pre-training)	0.9983 ± 0.0005
Fine-tuning CLS Tokens	0.9202 ± 0.0198
+ Inference parameters	0.9857 ± 0.0034
+ Remaining parameters	0.9953 ± 0.0013

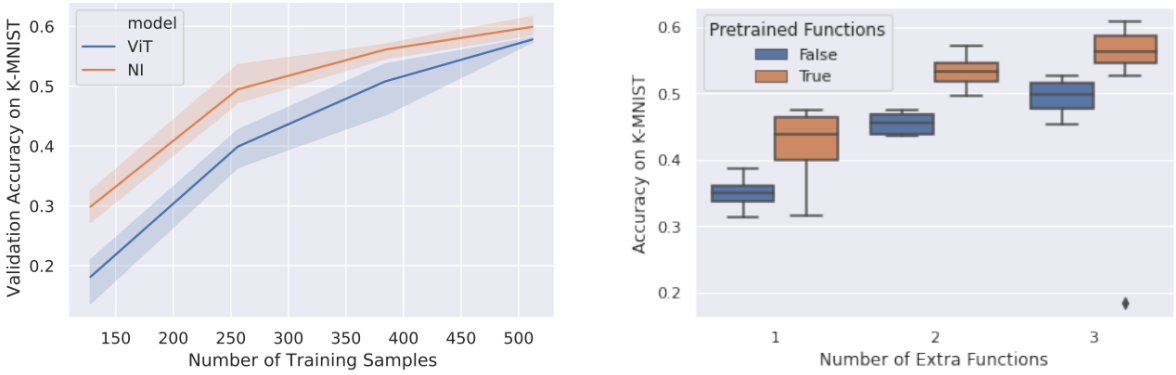
Table 4.2. *Mean Coefficient of Determination (R^2) and StdDev over 10 tasks after training various sets of parameters.*

Multi-Task Image Classification

They then consider four image datasets with the same sets of labels: **SVHN** [193], **MNISTM** [194], **MNIST** [159] and **KMNIST** [195]. They pre-train a NI model on the first three. They use as input 64 embedding vectors of image patches and 3 CLS vectors, each one representing a dataset. A classification head is attached to the output corresponding to each CLS vector and is used to predict the class of input image, when that image comes from the respective dataset.

After pre-training, they fine-tune on unaugmented KMNIST samples and compare the model's performance to the Vision Transformer's [40], which is a transformer model trained on image data. They first test whether the inductive biases implemented in NIs indeed offer an advantage when it comes to OOD generalization. As shown in figure 4.34a, the two models achieve the same performance on KNIST, given enough data, but NIs are faster to improve, suggesting that knowledge is stored in a way that decreases the number of updates that are needed to train the

model on a new distribution. They are also interested in the way knowledge is stored in the model, i.e. if invariant features are indeed stored in the functions' parameters. They therefore try re-initializing the parameters of the functions (signatures and codes) after pre-training and fine-tuning the resulting model. They observe that the model's performance when the pre-trained functions are used is much better revealing the importance of the information stored in them (figure 4.34b).



(a) *Fast Adaptation* **(b)** *Function Re-initialization*

Figure 4.34. *Figure (a) records the adaptation speed of a neural interpreter (orange) and a vision transformer (blue). Figure (b) presents the adaptation performance of the model for various numbers of re-initialized functions in the case where the pre-trained function vectors are used (brown) and in the case where they re-initialized (blue) [27].*

Moreover they are concerned with the question of whether new functions can be additionally installed and fine-tuned along with the pre-trained model. Results show that adding new functions does improve the model's performance, which means that the model has not overfitted to the existing functions.

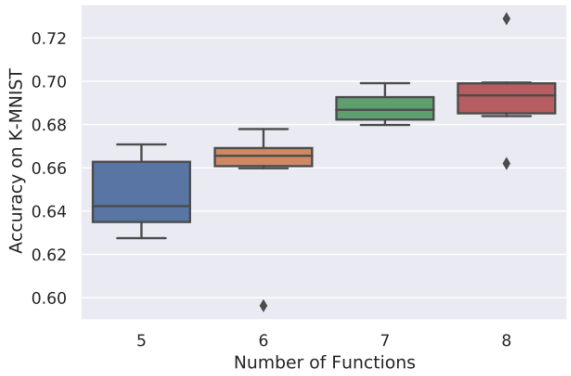


Figure 4.35. *Validation performance of NI models pre-trained with five functions and fine-tuned after the installation of additional ones, randomly initialized [27].*

OOD Generalization with Progressively Generated Matrices

Progressively generated matrices (PGMs) is a task that aims to test whether a model can generalize to OOD settings. A sample consists of 8 context and panels and 8 candidate answers as shown in figure 4.36. The model has to discover the pattern in the 8 given panels and choose among the 8 candidates for the 9-th piece.

Patterns involve logical relations and other linked to image attributes, e.g. shape, size, color, etc. 7 of the 8 datasets test the model's ability to systematically generalize, e.g. to relations not encountered during training. Rahaman et al. (2021) [27] hypothesize that the modular storage of knowledge will enable NIs to generalize better than models using a more rigid structure, like Vision Transformers [40].

They use a shallow CNN to create embeddings for each panel and use as input for the model sets of 10 vectors, 8 for the context panels, 1 for a candidate panel and 1 for a CLS token. The output corresponding to this token is used to produce a score for the respective candidate. The

scores for all candidates pass through a softmax layer and the model is optimized for the correct answers.

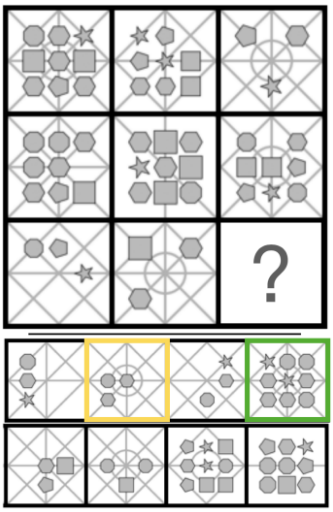


Figure 4.36. *A sample of the PGMs task. On top the 8 context panels are shown and below them are the 8 candidates [27].*

As indicated in table 4.2, NIs perform better than four out of six models in OOD generalization (test set) and are very competitive in in-distribution generalization (validation set).

4.13 Transformers with Competitive Ensembles of Independent Mechanisms

Similarly to Rahaman et al. (2021) [27], Lamb et al. (2021) [28] support the idea that modern NN architectures should be granted the freedom to decide how various subsets of their parameters are used. They contend that only parameter subsets that are deemed by the model to be relevant in a certain situation should be used, and others are considered irrelevant should not interfere and thus avoid being updated.

Specifically, they focus on the transformer model. Transformer models essentially apply the entirety of their parameter set to every position. According to Lamb et al. (2021) [28], providing the transformer architecture with the flexibility to decide when to use and when not to use the available neural modules would offer significant computational benefits and improve the ability of the model to generalize in OOD settings.

Regime	Neutral		Interpolation		Attribute P.		Triple P.		Triples		Extra.	
Model	**Val.**	**Test**	**Val.**	**Test**	**Val.**	**Test**	**Val.**	**Test**	**Val.**	**Test**	**Val.**	**Test**
WReN	63.0	62.6	79.0	64.4	46.7	27.2	63.9	41.9	63.4	19.0	69.3	17.2
VAE-WReN	64.8	64.2	-	-	**70.1**	**36.8**	64.6	43.6	59.5	**24.6**	-	-
MXGNet	67.1	66.7	74.2	65.4	68.3	33.6	67.1	43.3	63.7	19.9	69.1	18.9
ViT	73.3	72.7	**89.9**	67.7	69.4	34.1	67.6	44.1	73.8	15.9	**92.2**	16.4
NI	**77.3**	**77.0**	87.9	**70.5**	69.5	36.6	**68.6**	**45.2**	**79.9**	20.0	91.8	**19.4**

Table 4.2. *Accuracy of predictions of various models on several generalization tasks. The neural interpreter is compared to the wild relation network (WReN) [37], the variational autoencoder-WReN (VAE-WReN) [38], the multi-layer multiplex graph neural net (MXGNet) [39] and the vision transformer (ViT) [40]. The models are evaluated both on in-generalization performance (val.) and OOD generalization (test).*

They note that such structures cannot emerge through a standard training process alone, since fitting the training set usually does not require that this happens. Beaulieu et al. (2020) [24] (chapter 4.8.11) employ meta-learning to train their model. This results in their model learning to sparsely activate different parameter subsets depending on the input image, while avoiding having dead neurons. Training settings are indeed a way to introduce inductive biases (chapter 4.9.3). Another way is to implement an inductive bias by incorporating it into the model's architecture.

A common way of creating specialized modules in neural architectures is by establishing some form of competition between them (Parascandolo et al. (2017) [20], chapter 4.6.7; Goyal et al. (2019) [2], chapter 4.11). Lamb et al. (2021) [28] thus propose **transformers with competitive ensembles of independent mechanisms (TIMs)**, a transformer-based architecture that maintains distinct, parallel streams of computation that sparsely interact with each other, similarly to the RIMs model (chapter 4.11).

4.13.1 The Model

Group Linear Layers

In order to isolate the modules from each other they use **group linear layers (GLLs)**. GLLs use three-dimensional weight matrices $\mathbf{W} \in \mathbb{R}^{N_s \times D_{in} \times D_{out}}$, which can be viewed as N_s different two-dimensional weight matrices $\mathbf{W}_j$, $j \in \{1, \cdots, N_s\}$, that implement N_s linear layers. Each linear layer accepts as input a vector of dimension D_{in} and outputs a vector of dimension D_{out}.

$$GroupLinear(\mathbf{h}, \mathbf{W}, N_s) = [\mathbf{h}_j \mathbf{W}_j]_{j=1}^{N_s},\ \mathbf{h} \in \mathbb{R}^{N_s \times D_{in}} \tag{4.46}$$

Independent Mechanisms

Each mechanism uses the same weight sharing technique across positions used in the vanilla transformer architecture but no two mechanisms share weights with each other. Therefore, mechanisms can be viewed as parallel computational streams, each one implementing a smaller Transformer on its own. This means that, at every computational stage, there are N_s distinct representations for each element of the input sequence, each one belonging to a mechanism. Each mechanism performs its own self-attention over its representations and applies position-wise layer normalization, linear projections and FFN layers.

Competition for the Input

Each mechanism g_m, $m = \{1, \cdots, N_s\}$, uses a linear layer to compute a scalar for each input position $\mathbf{x}_i$, $i \in \{1, \cdots, T\}$. The scalars C_{mi} computed by each mechanism g_m, for each position $\mathbf{x}_i$, are used as inputs to a softmax layer that distributes $\mathbf{x}_i$ among the N_s mechanisms. The intuition and the method here resemble the distribution of inputs among functions in Neural Interpreters discussed in chapter 4.12. The scalars apart from routing the input vectors through the mechanisms also control how much each one will be updated as a result of its final contribution. This is a soft analogue of the competition method employed in RIMs (chapter 4.11), that completely deactivate irrelevant mechanisms.

Sparse Interactions

Lamb et al. (2021) [28] note that mechanisms in the real world may not be entirely independent and therefore, the neural structures modelling them should be allowed to sparsely exchange high-level information. They utilize an attention mechanism to allow for position-wise interaction between mechanisms. The attention module consists of 2 heads, each one with 32 units in order to constrain the flow of information and retain the independent character of the mechanisms.

An overview of the model is shown in figure 4.37.

Use of a TIM layer

A TIM layer can function as a drop-in replacement for a standard Transformer layer. Lamb et al. (2021) [28] state that they observed that keeping the first two and the last Transformer layers and replacing the intermediate ones with TIM layers lead to maximum improvements in performance for some tasks.

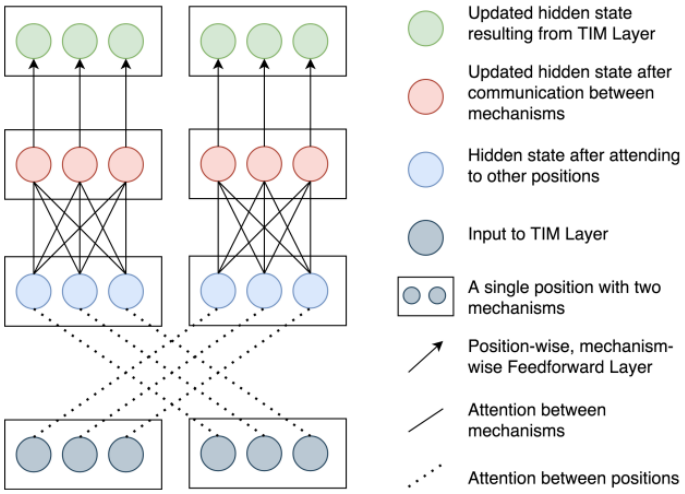


Figure 4.37. *Overview of a TIM model with three mechanisms and a two element input. Each mechanism maintains its own representation of each input element. First each mechanism performs self-attention using its representations of the input. Then, the mechanisms attend to each other's representations for each input element separately in order to exchange information. Finally a FFN layer is applied position-wise by each mechanism separately. Layer normalization and residual connections are also applied position-wise and mechanism-wise [28].*

The use of GLLs instead of linear ones reduces the number of parameters, but TIMs also introduce new ones. The replacement of a Transformer layer with a TIM layer typically reduces the total number of parameters by 30 to 40%.

4.13.2 Experiments

Image Generation

Lamb et al. (2021) [28] first test whether mechanisms specialize in a meaningful manner. They therefore replace layers of the image transformer [196] with TIM ones. The image transformer [196] is an image generating model based on the GPT-2 architecture. The training data Lamb et al. (2021) [28] synthesize consist of two images per sample, one coming from the MNIST [159] number dataset and the other from the CIFAR image dataset [197], whose samples vary from animals to vehicles. Since the two images, which are randomly sampled from the respective datasets, are clearly independent one expects, in the case of a TIM layers with 2 mechanisms ($N_s = 2$), to observe one mechanism focusing on image coming from one datasets and the other focusing on images coming from the other dataset.

ALGORITHM 4.6: *Equations of a Single TIM Encoder Layer*

Hyper-parameters: Number of mechanisms N_s, dimensionality of the keys, D_k, and values, D_v, number of heads for self-attention H, number of heads for inter-mechanism attention H_c.
$D_{mech} \leftarrow D_{model}/N_s$
$D_{FFN,m} \leftarrow D_{FFNN}/N_s$
Input: A vector $\mathbf{x}_i$ corresponding to the i-th input position

Perform Competition for Position i
$\mathbf{W}^c \in \mathbb{R}^{D_{mech} \times N_s \times 1}$
$\mathbf{C}_{\cdot;i} = \text{softmax}(\text{GroupLinear}(\mathbf{x}_i, \mathbf{W}^c, N_s)$

Mechanism-wise and Position-wise Self-Attention
$\mathbf{W}_S^Q, \mathbf{W}^K \in \mathbb{R}^{N_s \times D_{mech} \times HD_k}$
$\mathbf{W}_S^V \in \mathbb{R}^{N_s \times D_{mech} \times HD_v}$
$\mathbf{W}_S^O \in \mathbb{R}^{N_s \times HD_v \times D_{mech}}$
Compute query, key, and value for each mechanism and position
$Q_{s,i} = \text{GroupLinear}(\mathbf{x}_i, \mathbf{W}_S^Q, N_s)$
$K_{s,i} = \text{GroupLinear}(\mathbf{x}_i, \mathbf{W}_S^K, N_s)$
$V_{s,i} = \text{GroupLinear}(\mathbf{x}_i, \mathbf{W}_S^V, N_s)$
Perform Mechanism-wise Self-Attention
$\widetilde{\mathbf{h}}_i := \text{PositionAttention}(Q_{s,\cdot}, K_{s,\cdot}, V_{s,\cdot}, N_s H)[i]$
▷ result is shown for element in position i
$\widehat{\mathbf{h}}_i := \text{GroupLinear}(\widetilde{\mathbf{h}}_i, \mathbf{W}_S^O, N_s)$ ▷ Apply linear layer to each attention output separately
$\mathbf{h}_i := \text{norm}(\mathbf{x}_i + \mathbf{C}_{\cdot;i} \bigodot \widehat{\mathbf{h}}_i, N_s)$

Information Exchange Between Mechanisms
$\mathbf{W}_E^Q, \mathbf{W}_E^K \in \mathbb{R}^{N_s \times D_{mech} \times H_c D_k}$
$\mathbf{W}_E^V \in \mathbb{R}^{N_s \times D_{mech} \times H_c D_v}$
$\mathbf{W}_E^O \in \mathbb{R}^{N_s \times H_c D_v \times D_{mech}}$
Compute query, key, and value for each mechanism and position
$Q_{e,i} = \text{GroupLinear}(\mathbf{x}_i, \mathbf{W}_E^Q, N_s)$
$K_{e,i} = \text{GroupLinear}(\mathbf{x}_i, \mathbf{W}_E^K, N_s)$
$V_{e,i} = \text{GroupLinear}(\mathbf{x}_i, \mathbf{W}_E^V, N_s)$
Perform Mechanism-wise Self-Attention
$\widetilde{\mathbf{h}}_i := \text{MechanismAttention}(Q_{e,i}, K_{e,i}, V_{e,i}, N_s H_c)$
$\widehat{\mathbf{h}}_i := \text{GroupLinear}(\widetilde{\mathbf{h}}_i, \mathbf{W}_E^O, N_s)$ ▷ Apply linear layer to each attention output separately
$\mathbf{h}_i := \text{norm}(\mathbf{h}_i + \widehat{\mathbf{h}}_i, N_s)$

Mechanism-wise, Position-Wise FFN Layer Application
$\mathbf{W}_{FFN,1} \in \mathbb{R}^{N_s \times D_{mech} \times D_{FFN,m}}$
$\mathbf{W}_{FFN,2} \in \mathbb{R}^{N_s \times D_{FFN,m} \times D_{mech}}$
$\widetilde{\mathbf{h}}_i = \text{GroupLinear}(\sigma(\text{GroupLinear}(\mathbf{h}_i, \mathbf{W}_{FFN,1})), \mathbf{W}_{FFN,2})$
$\mathbf{h}_i := \text{norm}(\mathbf{h}_i + \widetilde{\mathbf{h}}_i, N_s)$

This is indeed depicted in figure 4.38 (right).

Moreover, the same model is given CIFAR-10 images, i.e. 32×32 coloured images in 10 classes, with 6000 images per class. One mechanism is found to specialize in the foreground and the other in the background of the image as shown in the left side of figure 4.38

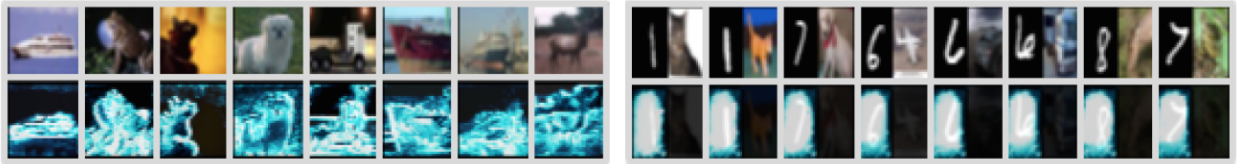


Figure 4.38. *A TIM model with 2 mechanisms is trained on CIFAR-10 images (upper left). One mechanism specializes in the foreground (shown here in bottom left) and the other in the background. Another TIM model with two mechanisms is trained on pairs of (MNIST and CIFAR) images (top right). The attention paid by the first mechanism to MNIST images is shown in the bottom right corner. This mechanism obviously specializes in MNIST digits [28].*

Speech Enhancement

Real-world speech signal are often accompanied by noise that is generated in the background. Traditional **speech enhancement** methods seek to improve the signal's quality by estimating the noisy part of it and then subtracting it from the signal.

Because of that, Lamb et al. (2021) [28] consider this task to be a good fit for TIMs, as a 2-mechanism TIMs model can naturally separate the two uncorrelated signals from each other. They train a Transformer with all of its layers, except the first two ones and the last one, replaced by TIM layers on the **deep noise suppression (DNS)** dataset. DNS is a corpus of 44 hours of clean and noisy speech signals, that are created by artificially injecting noise into clean ones.

They use the Perceptual Evaluation of Speech Quality (PESQ) score [197] for evaluating the quality of the output signal. They also test the model's ability to generalize to different noise distributions by evaluating it on the Voicebank test set [198].

As shown in table 4.3, TIM achieves state-of-the-art performance on the DNS dataset. It is shown by Lamb et al. (2020) that the signal is indeed separated on its speech and noise components. Moreover, TIM performs better than the transformer in OOD generalization, verifying their main hypothesis.

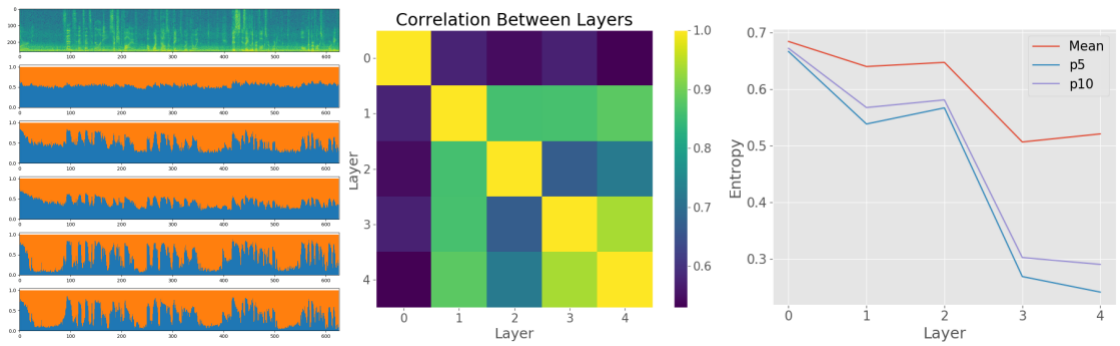


Figure 4.39. *A speech signal is shown on top of the left figure. The competition pattern is then shown for five successive TIM layers. The signal becomes clearer as one moves to from the input to the output layer. This increasing certainty about the competition winner is verified by the correlation matrix of competition over layers shown in the middle figure and by the progression of the competition's entropy from lower to higher, layers depicted in the right figure [28].*

Models Trained on DNS	Params (M)	DNS (PESQ)	VoiceBank (PESQ)
Noisy - no reverb	n/a	1.582	1.970
U-Net-MultiScale+	3.5	2.710	-
Conv-TasNet	5.1	2.730	-
PoCoNet	50.0	2.722	-
PoCoNet-SSL*	50.0	2.748	-
Transformer Baseline	6.1	2.727	2.517
TIM-NoComp ($N_s = 2$)	6.0	**2.754**	2.503
TIM-Comp ($N_s = 2$)	6.0	2.742	**2.575**
TIM-Comp ($N_s = 4$)	6.0	2.730	2.540

Table 4.3. *Models trained and evaluated on the DNS dataset. PoCoNet-SSL was trained on additional data. The TIM model is trained for $N_s = 2$ and $N_s = 4$. It is also trained without the use of the competition module for $N_s = 2$. It is compared to the original signal (Noisy - no reverb), the U-Net with a MultiScale+ cosine loss function (U-Net-MultiScale+) [41], the fully convolutional version of the time-domain audio separation network (Conv-TasNet) [42], the convolutional neural network with frequency-positional embeddings (PoCoNet) [43] and a transformer baseline. The TIM and the transformer models are also tested on OOD generalization to the VoiceBank test set.*

Language Modeling and GLUE Benchmark

TIMs are also tested on NLP tasks. Lamb et al. (2021) [28] replace all layers of a BERT model except the first two and the last one with TIM layers with two mechanisms. The model is pre-trained with a LM task on BookCorpus [137] and the Wikipedia corpus as is BERT. It is then fine-tuned on the GLUE dataset.

TIM is found to perform better than BERT, both in terms of performance (table) and in terms of stability of training, as they report observing the variability of the scores for different initializations of the model, i.e. different seeds, to be smaller than BERT's.

Result	BERT	BERT-130M	TIM-All-Layers	TIM-NoComp	TIM-Comp
TIM Layers	0/12	0/12	12/12	9/12	9/12
Parameters	110M	130M	110M	130M	130M
Competition	No	No	No	No	Yes
Valid-NLL	2.096	2.040	2.112	2.033	**2.027**
MNLI-M	84.93 ± 0.15	85.37 ± 0.29	84.19 ± 0.34	**85.89 ± 0.17**	85.28 ± 0.22
MNLI-MM	84.91 ± 0.18	85.28 ± 0.27	84.55 ± 0.15	**85.80 ± 0.07**	85.17 ± 0.18
QNLI	91.34 ± 0.21	91.84 ± 0.32	91.37 ± 0.59	91.78 ± 0.14	**91.97 ± 0.20**
SST-2	92.88 ± 0.33	92.75 ± 0.26	92.52 ± 0.56	92.75 ± 0.13	**92.97 ± 0.25**
STS-B	89.43 ± 0.25	89.34 ± 0.15	88.20 ± 0.32	88.52 ± 0.28	**89.63 ± 0.05**

Table 4.4. *Compare the perplexity on the validation set (Valid-NLL) and the performance one various GLUE tasks of two BERT models with different sizes, and three TIM models of various configurations.*

CATER Occluded Object Tracking

The diagnostic dataset for **compositional actions and temporal reasoning (CATER)** [199] is a spatio-temporal reasoning video task that involves using the frames of a video as input to predict an object's final location in the scene. The object might be occluded during part of the video, so the model must take into consideration the interactions between objects in the scene, i.e. a ball might be places under a cup and then the cup might be moved.

The task is turned into a classification task, as the image areas are separated by 6×6 grid, and the model is asked to choose the rectangle where it believes the object is found. The features used by the neural models are created after using the frames as input to a ResNet block. TIMs perform between than transformer model and their performance improves as the number of mechanism increases.

Model	**Top 1** %	**Top 5** %
LSTM	67.4	85.8
Transformer	68.7	81.7
TIM-COMP $N_s = 2$	68.4	85.7
TIM-COMP $N_s = 4$	71.0	**87.3**
TIM-COMP $N_s = 8$	**71.1**	87.2

Table 4.5. *Comparison on CATER Object Tracking of the Top-1 and Top-5 accuracy of Transformers with TIM.*

Chapter 5

Transformers with Competitive Attention Heads and FFNNs

5.1 An Alternative Idea on the Dimensionality of Representations

5.1.1 Advantages of High-Dimensional Representations

Most of the works that were presented in chapter 4 seek to implement the inductive biases of modularity and independent mechanisms, and high-level variables (chapters 4.3, 4.6.5 and 4.9.5) by structurally segmenting the network into individual neural modules and reducing the dimensionality of the respective representations (chapters 4.6.7, 4.11, 4.12 and 4.13). In fact, Goyal et al. (2019) [2] and Rahaman et al. (2021) [27] explicitly refer to the process of dimensionality reduction as a means to promote the use of high-level variables by their models.

Nevertheless, there is neuroscientific evidence suggesting that the neural representations of visual objects are stored in high-dimensional distributed forms across various regions within the human brain [200], and that the connections between neural areas resemble small-world type graphs, which makes the boundaries between them rather indistinguishable [71]. Karneva (2009) [201] contends that the enormous quantity of neurons and synapses in the human brain suggests that employing **high-dimensional representations** for information storage and processing might be necessary in order to perform brain-like functions. He believes that the incredible robustness of neural representations to various transformations and noise additions stems from **redundancies** in the way knowledge is stored in the human brain, just like the way redundancy is employed to ensure the integrity of transmissions in telecommunications. He notes that the proportion of errors that is acceptable, in the sense that it allows the signal to be successfully retrieved afterwards, increases with the dimensionality of the signal, further strengthening the argument in favour of high-dimensional representations.

Karneva (2009) [201] also discusses the matter of how information may be distributed in each of these high dimensional representations to ensure such robustness. As mental happenings cause various neural areas to simultaneously activate, Karneva (2009) [201] suggests that information must be **distributed evenly** among each vector's positions. In a setup like the one described above, the representations become robust to events of localized destruction of information.

Finally he considers the use of **randomness** by the human brain to be necessary, when constructing a new object, and is, as he claims, imposed by the unique initialization of every human brain. He therefore notes that compatibility between human heads is not found in the representations themselves, which may differ from brain to brain, but in the relationships between representations used by each single brain and in the way these are manipulated, consciously or un-

consciously. He calls these protocols a **cognitive code**, and thinks that a significant step towards human-like intelligence would be the discovery of the rules that make up this code.

5.1.2 A 10000-Dimensional Example

He then provides a concrete example of his argument, by discussing the properties of a 10,000-dimensional binary vector. The choice of a binary vector is made to simplify the argument and to show that the advantages of using high-dimensional representations originate exclusively from their dimensionality.

In a 10,000-bit vector space, two points, A and B, are deemed **similar** if their Hamming distance, i.e. the number of digits of A one must flip to obtain B, is somewhat closer than 5,000 bits, which is the mean distance between two random vectors of the vector space. Two vectors that have an approximate distance of 5,000 are considered to be **unrelated**. More specifically, only a millionth of the space is closer than 0.476, i.e. 4,760 bits, and only a thousand millionth closer than 0.47. Similarly for two vectors being further than 0.524 and 0.53 from one another accordingly. Therefore it is practically impossible for two vectors that are randomly instantiated to represent two distinct entities to end up being similar. This is indicative of the high robustness of the retrieval system to random noise, i.e. when provided with a noisy version of a vector it is very likely that the system will be able to retrieve the original vector, as it will most likely be the closest instantiated vector to the given sample. In fact, each vector has a large "private" neighborhood as the portion of vectors that are found at a distance smaller than 0.33 from it is minuscule when compared to the entire vector space. This neighborhood can be used to instantiate vectors that are similar to one or more representations. These representations, on the other hand, don't have to be similar to each other, e.g. the distance between two vectors, **A** and **C**, that are similar to a vector **B**, that is 2,500 bits away from each one of them can very well be 5,000 bits.

5.1.3 Computing with High-dimensional Vectors

He further discusses the matters of hyper-dimensional memory and arithmetic. He proposes the use of an **auto-associative memory**, i.e. one that stores each item in a position whose address is equal to the value of this item. One can then retrieve an item **X** even when probing the address with a noisy version of **X**, **X**'.

Hyper-dimensional arithmetic uses the same set of basic operations used in standard vector arithmetic. But Karneva (2009) [201], also introduces a few modifications, e.g. the representation of sets of vectors by their sums, of pairs of vectors by vector multiplications, of sequences by permuting sums, etc.

5.2 Modeling High-Level Variables with Standard Neural Networks

The idea of independent mechanisms is indeed promising, but the research community still struggles to successfully implement the related notions and integrate them into neural models. There are various questions that remain unanswered. The most important one is the issue of discovering the underlying causal variables from raw, low-level data. Bengio et al. (2020) [25] (chapter 4.10), discover the two causal variables that have been transformed by a rotation matrix, using a meta-learning objective that is based on the high adaptation speed of valid causal models. But it is yet unclear how to solve this problem in real-world settings, where the number of underlying variables and the complexity of each one are unknown.

As noted in chapter 4.9, Goyal and Bengio (2022) [51] contend that NNs excel at performing System 1 - type tasks, and they seek to model System 2 - type functions by introducing a set of inductive biases. They propose several architectures (chapters 4.11, 4.12, 4.13) as well as training processes (chapter 4.10) in an effort to implement the notions of independent causal mechanisms, modularity, high-level variables and specialization. Yet, it is currently uncertain whether systems modeling high-level variables and interactions between them, as those that occur during conscious processing, can be implemented using the same architectures and training protocols as the ones used for processing low-level variables. Entirely new architectures with the ability to manipulate object-level variables may have to be engineered, as well as interfaces between them and the neural software dedicated to the modeling of low-level variables and processes.

It must be noted that the use of the ideas of independent causal mechanisms and modularity as starting points in the process of designing systems that are able to perform System 2 - type functions is certainly considered promising and worthy of further investigation. What is questioned is whether one can rely solely on the standard neural network framework to play the role of the building block for these kinds of systems. As discussed in chapter 5.1, implementing these ideas with neural network architectures often results in the reduction of the dimensionality of the internal representations which contradicts the intuition related to high-dimensional representations suggested by Karneva (2009) [201]. Moreover, findings that brain neurons are organized in small-world networks indicate that neural modules are interconnected in complex ways and therefore modeling them by structurally segmenting the network might not be ideal.

5.3 Proposed Methods

We thus take a step back and stick to the inter-connectivity allowed by standard transformer-based architectures while, at the same time, providing the model with the flexibility to update sets of parameters that perform different functions independently from one another.

5.3.1 Competitive FFNNs

First, we propose the replacement of the FFNNs that follow the attention mechanisms in transformer-based architectures with series of N_f parallel FFNNs. In order to promote the specialization of each FFNN of each model layer we propose to distribute the input elements among the FFNNs via an attention mechanism. Attention essentially implements a form of competition among the FFNNs, is inspired by Parascandolo et al. (2017) [20] (chapter 4.6.7) and applied in a fashion similar to Lamb et al. (2021) [28] (chapter 4.13).

Specifically, adopting the terms used by Rahaman et al. (2021) [27] (chapter 4.12), each FFNN of a layer l, $l \in \{1, \cdots, L\}$, F_{li}, $i \in \{1, \cdots, N_f\}$, is assigned a signature vector $\mathbf{f}_{li} \in \mathbb{R}^{D_f}$. A new FFNN, called inference FFNN, InfF_l, uses the output of the attention mechanism of the same layer l at each position j, $\mathbf{h}_{lj}$ to produce a corresponding key vector: $\mathbf{k}_{lj} = \text{InfF}_l(\mathbf{h}_{lj}) \in \mathbb{R}^{D_f}$. The key and the signature vectors belong to the same semantic space $\mathcal{S}$. Compatibility values between each element $\mathbf{h}_{lj}$, that has already been processed by the previous attention layer, and the FFNNs of the corresponding layer l, $\{F_{li}\}_{i=1}^{N_f}$, are computed as follows:

$$Cf_{lij} = \text{softmax}_i(\frac{\mathbf{f}_{li}\mathbf{k}_{lj}}{\sqrt{D_f}}) \tag{5.1}$$

where the softmax is computed along the axis symbolized with i, which corresponds to the FFNNs of the l-th layer.

After the FFNNs of the l-th layer have been applied to $\mathbf{h}_{lj}$, the weighted sum of their outputs

is computed to create the module's final output:

$$\mathbf{o}_{lj} = \sum_{i=1}^{N_f} Cf_{lij} F_{li}(\mathbf{h}_{lj}) \tag{5.2}$$

5.3.2 Competitive Attention Heads

Similarly to the case of competitive FFNNs, it is also reasonable to seek to establish some form of competition between the attention heads of each model's layer. As shown by Voita et al. (2019) [16] (chapter 4.4.2) and Clark et al. (2019) [17] (chapter 4.4.3), different attention heads tend to specialize in different positional, semantic or syntactic relations. Nevertheless, as proved by Cordonnier et al. (2021) [13] (chapter 3.10), attention heads are severely underutilized. We hypothesize that, by forcing them to compete for the input sequence's positions, they will begin capturing sufficiently distinct aspects of natural language, leading to improvements in both in-distribution and OOD generalization abilities of the model, as shown by Lamb et al. (2021) [28] in the case of BERT.

Again, one can employ an attention-based mechanism to implement the aforementioned competition. For each head H_{li}, $l \in \{1, \cdots, L\}$, $i \in \{1, \cdots, H\}$, a signature vector $\mathbf{a}_{li} \in \mathbb{R}^{D_a}$ is instantiated to play the role of the query vector. For the representation of each output of the previous module, which is also the input to the current attention layer, and is found in the j-th position of the input sequence, $\mathbf{h}_{lj}$, a key is computed by a separate FFNN, InfA_l: $\mathbf{k}_{lj} = \text{InfA}_l(\mathbf{h}_{lj}) \in \mathbb{R}^{D_a}$. The key and the signature vectors belong to the same semantic space $\mathcal{A}$. Compatibility values between each input element $\mathbf{h}_{lj}$, and each attention head H_{li}, $\{H_{li}\}_{i=1}^{H}$, are computed as follows:

$$Ch_{lij} = \text{softmax}_i(\frac{\mathbf{a}_{li}\mathbf{k}_{lj}}{\sqrt{D_a}}) \tag{5.3}$$

where the softmax is taken along the axis symbolized with i, that corresponds to the heads of the attention mechanism of the l-th layer.

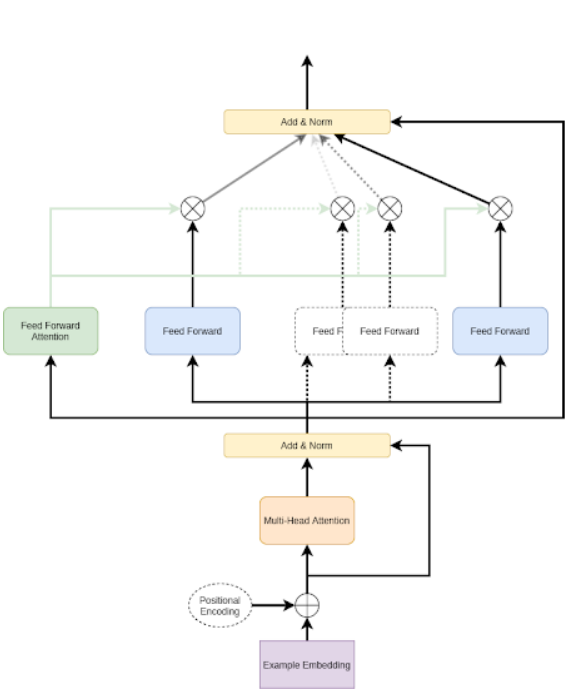
The output of a head H_{li} for the j-th position is given by $\mathbf{o}_{lij} = Ch_{lij}\tilde{\mathbf{h}}_{lj}$, which essentially is a scalar-vector multiplication, and where $\tilde{\mathbf{h}}_{lj}$ would be the output of the attention head had there not the competitive attention heads mechanism been used.

Similarity to the DropHead Algorithm

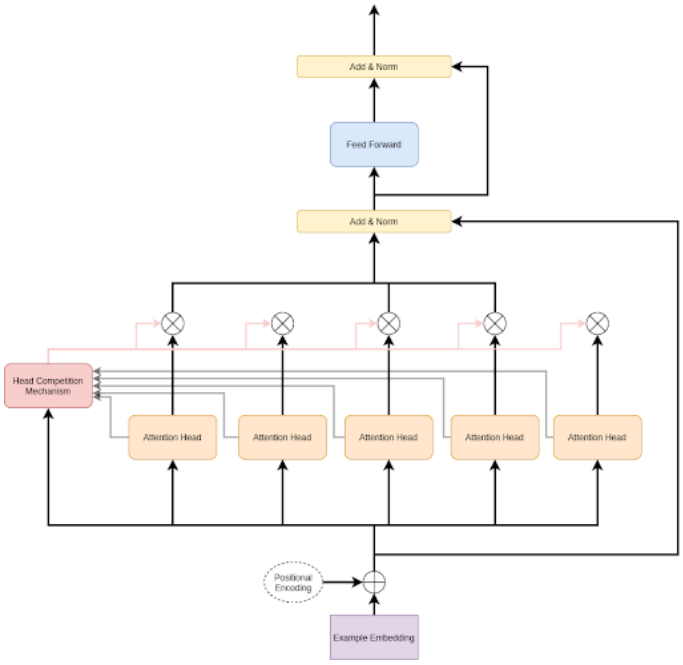
The competitive attention heads method resembles drophead proposed by Zhou et al. (2020) [14] and discussed in chapter 3.11 in that a scalar is used to modulate the outputs of entire attention heads, but differs in two ways. First, the scalars in the case of the drophead method are binary, and thus only activate or deactivate heads whereas, in the case of the competitive heads algorithm, the scalars are real numbers that receive their values anywhere inside the interval (0, 1). One could thus say that the latter is a soft analogue of the first. Second, and most important, the drophead method randomly drops heads, while the competitive attention heads method does so in a deterministic manner, based on the input of the corresponding attention mechanism.

5.4 Experiments

We apply the competitive attention heads technique (CAH), discussed in chapter 5.3.2, to the Transformer model and to a BERT-based model. In this thesis we only experiment with in-distribution generalization, i.e. the samples of the test set and the ones of the training set originate from the same distribution, leaving OOD generalization for future work. Lamb et al. (2021) [28]



(a) *Overview of the competitive FFNNs method*



(b) *Overview of the competitive attention heads method*

Figure 5.1. *Overview of the proposed methods*

showed that the implementation of the associated inductive biases leads to improvements in both the in-distribution and OOD generalization capabilities of the model.

5.4.1 FAIRSEQ

The methods are written with pytorch [202] and are integrated into existing model implementations available in the FAIRSEQ library [53]. FAIRSEQ is a library that maintains code for a variety of models that perform text generation in the contexts of NMT, summarization, NLM, etc. Each neural structure in the hierarchical pipeline, e.g. the attention mechanism, has its source code placed in a separate file and interacts with the structures that are lower in the hierarchy by considering them as black boxes. This enables us to modify an architectural component without affecting the rest of the neural structures.

5.4.2 Application to the Transformer Model

We first apply the CAH method to the attention mechanisms of the transformer architecture [1], i.e. the encoder self-attention, the decoder self-attention and the encoder-decoder attention. The model has $L = 6$ layers, the dimension of its internal representations is equal to $D_{model} = 512$, the dimension of the hidden layers of the FFNNs is $D_{ffnn} = 1024$ and InfA_l, $l \in \{1, \cdots, L\}$, are FFNNs, each with one hidden layer of dimension D_{InfA}.

The transformer is trained on a NMT task, and specifically on WSLT_14 [30], which is a collection of datasets that was used in tasks of the Ninth Workshop on Statistical ML. It involves four tasks: a news translation task, a quality estimation task, a metrics task and a medical text translation task. The variant used here is the English to German one (en-de).

We train a model for every possible combination of use (marked with cross) or no use (blank) of the method in each of the three types of attention of a transformer model. Though the number of parameters varies between models because of the number of attention types the CAH method

is applied to, CAH only introduces a few new parameters and all models have roughly 41 million parameters. Zhou et al. (2020) [14] contend that drophead enabled them to increase the number of heads leading to improved results. We thus report results for both a 4-headed and an 8-headed model, which are presented in table 5.1. We report the perplexity metric (PPL) (chapter 3.3.2) for the training and test set and the BLEU metric [203] (chapter 3.6) for the test set.

Enc. SA	Dec. SA	Enc.-Dec. Attn	# of Heads	PPL Training	PPL Test	BLEU Test
			4	5.22	5.45	33.25
+			4	4.82	5.41	33.61
	+		4	5.14	5.46	33.28
+	+		4	4.66	5.42	33.54
		+	4	5.30	5.52	33.32
+		+	4	4.95	5.46	33.20
	+	+	4	5.13	5.44	33.46
+	+	+	4	4.82	5.43	33.47
			8	5.24	5.48	33.16
+			8	4.78	5.53	32.83
	+		8	5.49	5.63	32.41
+	+		8	4.64	5.55	32.89
		+	8	5.33	5.59	33.19
+		+	8	5.15	5.69	32.68
	+	+	8	5.14	5.52	33.18
+	+	+	8	4.48	5.51	33.28

Table 5.1. *Application of the CAH method to the three attention mechanisms of a transformer model with $L = 6$, $D_{model} = 512$, $D_{ffnn} = 1024$, $D_a = 24$, $D_{InfA} = 128$. Enc. SA stands for the encoder self-attention mechanisms, Dec. SA for the decoder self-attention mechanisms and Enc.-Dec. Attn for the encoder-decoder attention mechanisms. A cross (+) is used to symbolize the use of the CAH method at the respective mechanism and a blank to symbolize its absence. The perplexity on the training set (PPL Training) and the test set (PPL Test) and the BLEU metric on the test set are reported. The model was trained for 15 epochs.*

Apart from a slight increase in the BLEU metric in the cases of application of the method to the encoder self-attention alone and to the self-attention mechanisms of both the encoder and the decoder for a 4-headed model, no other significant improvement is evident.

Ablation Study On The Dimensionality

An ablation study is performed on the effects of the signature vector dimension, D_a, and the hidden dimension of the inference FFNN, D_{InfA}, on the model's performance. No clear correlation between the quantitative results and the dimensionality of the matching and inference system seems to exist (table 5.2).

Deactivating The Least Compatible Head

We then attempt fully deactivating the head that receives less attention at each position, i.e. the inner product of its signature vector with the corresponding element of the input sequence is the smallest among heads of the same layer, and apply softmax to calculate compatibility values for the rest. This is a combination of ideas from Goyal et al. (2019) [2] and Rahaman et al. (2021) [27]. The experiment is performed on a 4-headed transformer model. Again no clear signs of improvement are shown (table 5.3).

D_a	D_{InfA}	PPL Training	PPL Test	BLEU Test
12	64	4.83	5.46	33.27
12	128	4.81	5.44	33.38
12	256	4.75	5.37	33.71
24	64	4.76	5.37	33.46
24	128	4.82	5.43	33.61
24	256	4.78	5.43	33.55
48	64	4.85	5.44	33.46
48	128	4.84	5.45	33.38
48	256	4.76	5.41	33.53

Table 5.2. *Ablation study on the dimension of the signature vectors of the attention heads, D_a, and the dimension of the hidden layer of the inference FFNN, D_{InfA}. The model is a transformer with $L = 6$, $D_{model} = 512$, $D_{ffnn} = 1024$, $H = 4$ and the CAH method is only applied to the encoder's self-attention mechanism.*

Enc. SA	Dec. SA	Enc.-Dec. Attn	PPL Training	PPL Test	BLEU Test
			5.22	5.43	33.25
+			4.85	5.45	33.51
	+		5.16	5.45	33.41
+	+		4.73	5.49	33.23
		+	5.33	5.54	33.19
+		+	5.02	5.52	33.25
	+	+	5.19	5.51	33.33
+	+	+	4.83	5.45	33.42

Table 5.3. *Application of the CAH method to the three attention mechanisms of a transformer model with $L = 6$, $D_{model} = 512$, $D_{ffnn} = 1024$, $D_a = 24$, $D_{InfA} = 128$, $H = 4$. After the inner product inside the softmax of the equation 5.3 is computed for the signature functions of all attention heads and the key corresponding to an element located in a certain position i, the head with the smaller respective product is deactivated for the position i and the rest of the inner products are used to compute compatibility values for the remaining heads. The perplexity on the training set (PPL Training) and the test set (PPL Test) and the BLEU metric on the test set are reported. The model was trained for 15 epochs.*

Training Inference Modules and Rest of The Model Separately

It is possible that training the matching and inference modules and the rest of the model parameters simultaneously leads to the model overfitting the modules. We therefore try training the modules, that are made of the inference FFNNs and the head signatures, separately from the rest of the model. For this purpose we split the training dataset into two parts, a large one consisting of 149 thousand examples and a smaller one with 11.2 thousand examples. In order to choose the sizes of the two datasets we initially split the dataset following equation 2.55 (ratio of parameter numbers is approximately 1:40) and then started gradually increasing the number of examples contained in the smaller part until performance on the validation set began to drop. We employ two modes of training:

- Sequential training: we keep the inference modules' parameters frozen in the first 15 epochs of training and train the rest of the model using the big part of the dataset. Then we train only them on the small part dataset for 5 more epochs, while freezing the rest of the net.

- Iterative training: we keep the parameters of the inference modules frozen for one epoch while training the rest of the network on the big part of the dataset. In the next epoch we train them only on the small part. We repeat this process performing 30 training epochs in total.

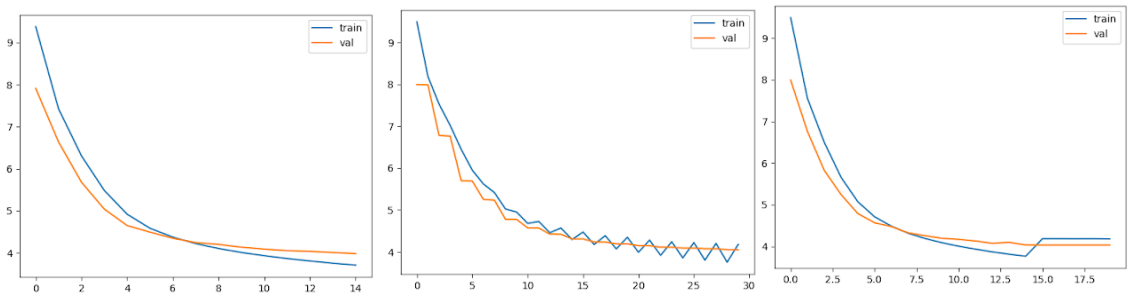
Notice that when a model is trained in an iterative fashion, it is a trained on each single sample 15 times in total, which is equal to the number of epochs that was used in the previous experiments. The reason why we don't show results for 15+15 epochs of sequential training, but only for 15+5, is because we noticed that using 30 epochs in total did not lead to considerable performance improvements.

We tried applying the method to each one of the self-attention mechanisms separately and on both of them at the same time. The results are reported in table 5.4. The experiments showed no sign of improvement. On the contrary, a slight drop in performance is observed in most of the cases.

Enc. SA	Dec. SA	Training Mode	PPL Training	PPL Test	BLEU Test
+		Jointly	4.82	5.41	33.61
	+	Jointly	5.14	5.46	33.28
+	+	Jointly	4.66	5.42	33.54
+		Sequential	5.35	5.77	32.10
	+	Sequential	5.39	5.71	32.41
+	+	Sequential	5.33	5.81	32.52
+		Iterative	5.29	5.76	32.24
	+	Iterative	5.53	5.76	32.27
+	+	Iterative	5.27	5.79	32.22

Table 5.4. *The transformer model is trained in three distinct manners: jointly, sequential and iterative. The CAH method is applied each one of the self-attention mechanisms separately and on both of them at the same time. The perplexity on the entire training set (all 160.2 thousand examples) (PPL Training) and the test set (PPL Test) and the BLEU metric on the test set are reported. The model was trained for 15 epochs.*

We also include a few plots which offer valuable insights into the training process of the Transformer model that has the CAH method applied to its encoder's self attention mechanism. We plot the respective learning curves in figure 5.2, as well as the evolution of the BLEU metric on the validation set as training proceeds in figure 5.3.

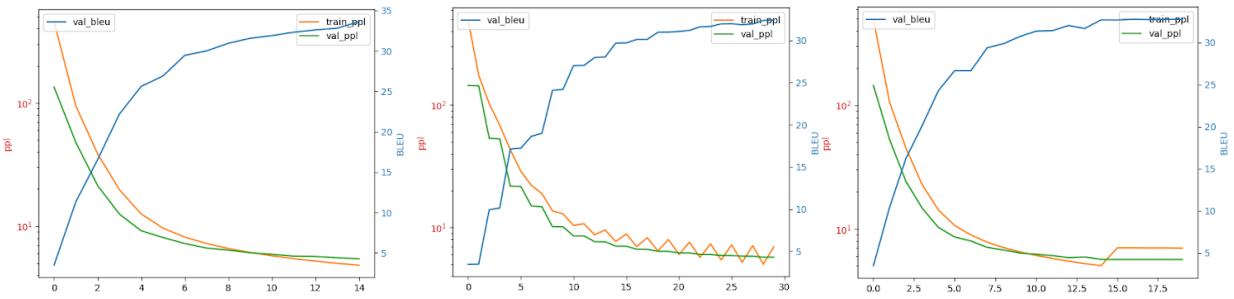


(a) *Trained Jointly* **(b)** *Iterative Training* **(c)** *Sequential Training*

Figure 5.2. *Learning curves for the three modes of training. The loss is computed on the dataset used in the last epoch. This is the reason for the scissor-like effect observed in* ***(b)*** *and the sudden increase observed at epoch 15 in* ***(c)*** *[5].*

During iterative training, epochs that are used to train the matching modules don't seem to contribute much to the increase in the BLEU metric as the corresponding curves are near to being horizontal. This of course might seem reasonable, given that only a small subset of the data is used in those epochs. The BLEU metric observed during the sequential training process also seems stagnant after the fifteenth epoch, when the inference modules alone are trained.

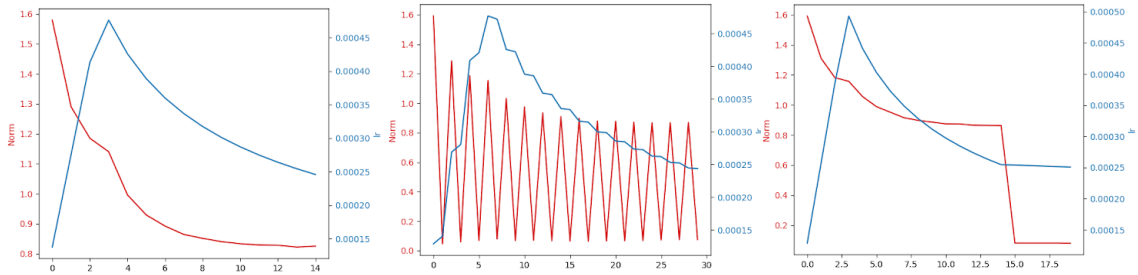
The learning rates and the gradient norms are also plotted in figure 5.4. Moreover, in order to gain a better understanding of how our modification affects the internal workings of the model, we



(a) *Trained Jointly* **(b)** *Iterative Training* **(c)** *Sequential Training*

Figure 5.3. *Evolution of perplexity on the training and validation sets and of the BLEU metric on the validation set for the three modes of training. They are computed on the dataset used in the last epoch. This is the reason for the scissor-like effect observed in figure* ***(b)*** *and the sudden increase observed at epoch 15 in figure* ***(c)*** *[5].*

also plot the evolution of the norms of the attention heads' signatures during training.



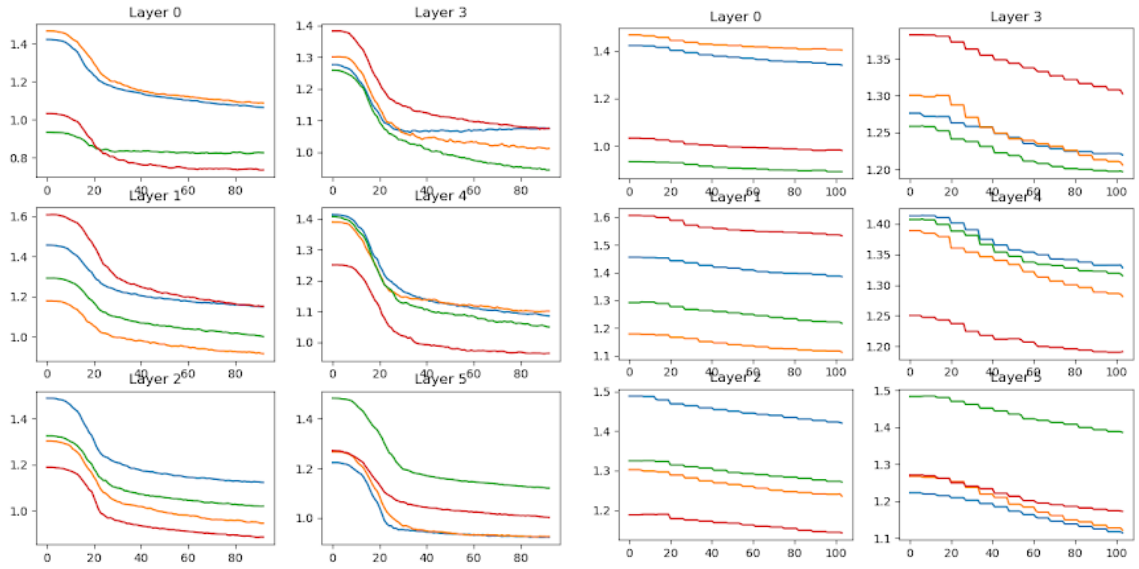
(a) *Trained Jointly* **(b)** *Iterative Training* **(c)** *Sequential Training*

Figure 5.4. *Evolution of gradient norm (red) and the learning rate (blue) during training for the three training modes. The gradient norm decreases in epochs where the small subset of the dataset is used because of the smaller number of parameters that are trained in these epochs. This is the reason for the scissor-like effect observed in figure* ***(b)*** *and the sudden fall observed after the fifteenth epoch in figure* ***(c)****. The linear increase in the learning rate in the beginning is due to a warm-up period that is usually employed to stabilize training [5, 29].*

To obtain a clearer image of the dynamics within the model we train it for 100 epochs, at which point all model variations have started to overfit, and then plot the norms of the heads' signatures for all heads of the model. When all the model's components are trained jointly, the norms of the signatures slowly decrease in the first two to three epochs of training, a behaviour that is followed by a sharp fall and then a return to a smaller rate of decrease (figure 5.5a). A slower decrease of the norms of the signatures occurs when the model is trained in an iterative manner (figure 5.5b), which is expected given the use of a smaller dataset for this purpose. A slightly sharper fall is observed during the second part of the sequential training process (figure 5.6). Therefore, in all cases the inference modules are indeed trained and continue changing after the respective models have begun to overfit.

Testing For Head Specialization

If the heads do indeed specialize, then it is reasonable to expect from different heads to focus on different word categories. For example, a head could focus on words that belong to a particular part of speech. It is also possible for a head to specialize in words that are semantically connected, e.g. small real-world objects constitute, for example, a semantic category.



(a) *Trained Jointly* **(b)** *Iterative Training*

Figure 5.5. *Evolution of the norms of the head signatures for all heads of the model, presented separately for each layer, for the cases where all model components are trained jointly* ***(a)*** *and in an iterative manner* ***(b)*** *[5].*

An easy way to test whether heads do indeed specialize is to plot the mean of each one's compatibility values for each position of the input sentence separately. The distribution of word appearance differs from sentence position to sentence position. For example, pronouns and prepositions are known to be found in the first position of a sentence more often than in the second one. We thus plot the evolution of the average compatibility value over the examples of each training batch (batch_size=128) throughout the training process for each one of the first five sentence positions separately, excluding the $< start >$ token.

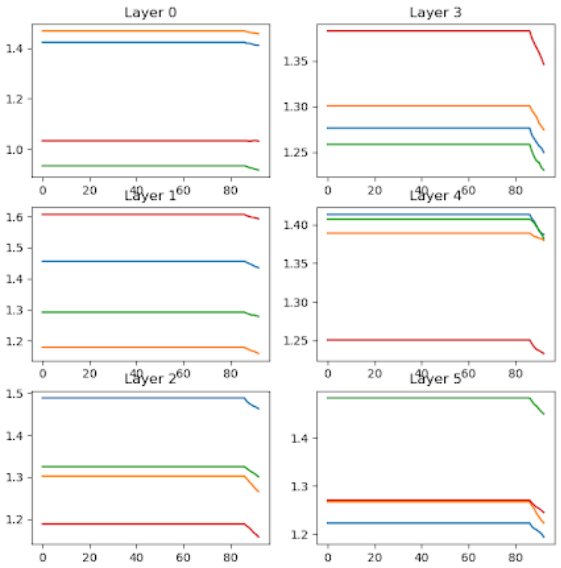
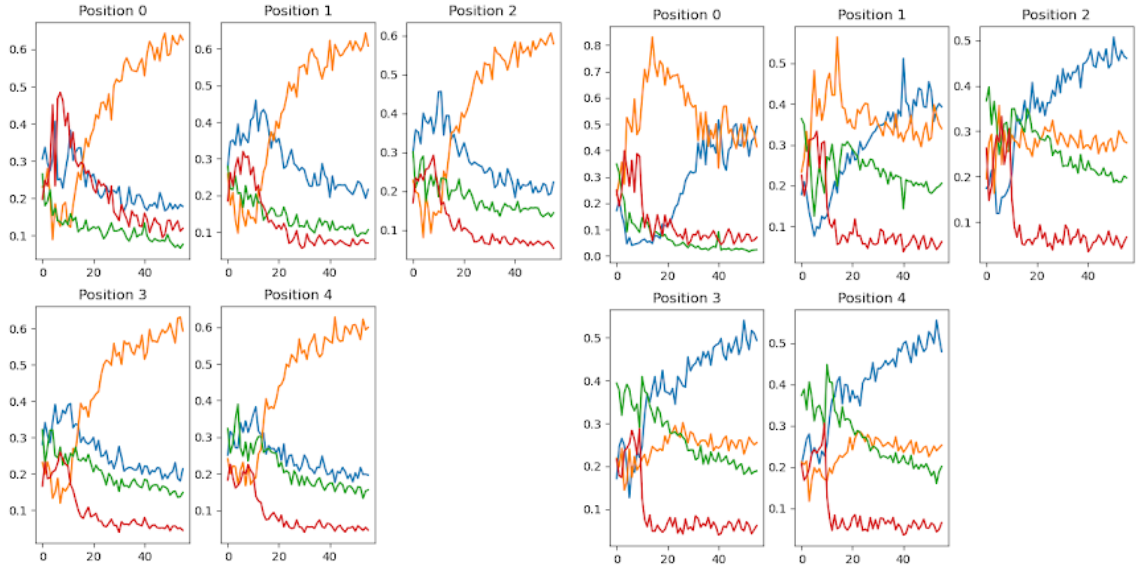


Figure 5.6. *Evolution of the norms of the head signatures for all heads of the model, presented separately for each layer, for the cases where all model components are trained in a sequential manner [5].*

For the first mode of training, where the entire model is trained in every epoch, we plot the average compatibility values for the third, the fourth ad the fifth layer of the encoder, which we consider to be representative (figures 5.7 and 5.8). We observe that the distributions of the compatibility values do not change much from position to position, regardless of the head being considered. Obviously, the heads do not specialize as they are expected to. In fact, a single head is shown to dominate all positions in the third layer.

A possible explanation for this observation is that the norms of the signals exiting the attention mechanism may cancel out the differences between the contribution values assigned to the heads. We thus plot the evolution of their norms alongside the resulting norms after the signals have been



(a) *Third Layer* **(b)** *Fourth Layer*

Figure 5.7. *Evolution of the mean compatibility values assigned to each head across a batch of samples for each of the first five positions of the input sentence during training. The values for the heads of the third* ***(a)*** *and the fourth* ***(b)*** *layer are shown [5].*

multiplied with the respective scalar contributions. We observe that this is indeed the case for the third (figure 5.9) and the fifth layer (figure 5.11).

What is very interesting in the case of the third layer, for example, is that the norm of the input signal to the head that corresponds to the yellow colour is much smaller than the norms of the others but, when the signal is multiplied with the respective scalar contribution, it ends up having a norm comparable to the rest of the outputs. The transformer mechanism thus seems to partly cancel the competitive heads modification.

In order to discover the actual impact of our method to the model we plot the norms of the outputs of the attention heads of the vanilla transformer architecture. The plots for its third, fourth and sixth layers are shown in figures 5.12 and 5.13. The norms differ from each other as do the norms of the weights of the projection layers that process each of these outputs.

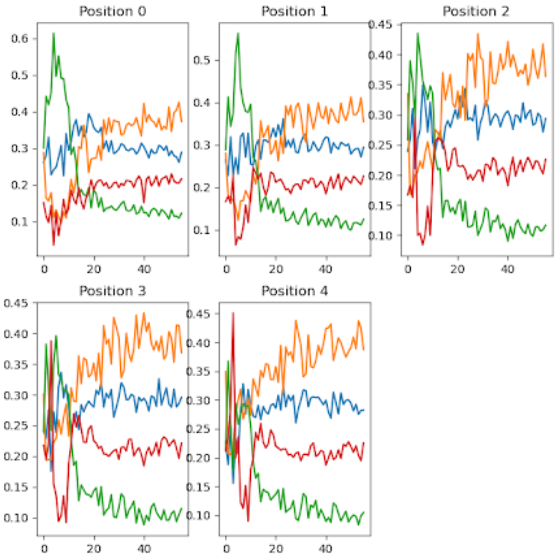
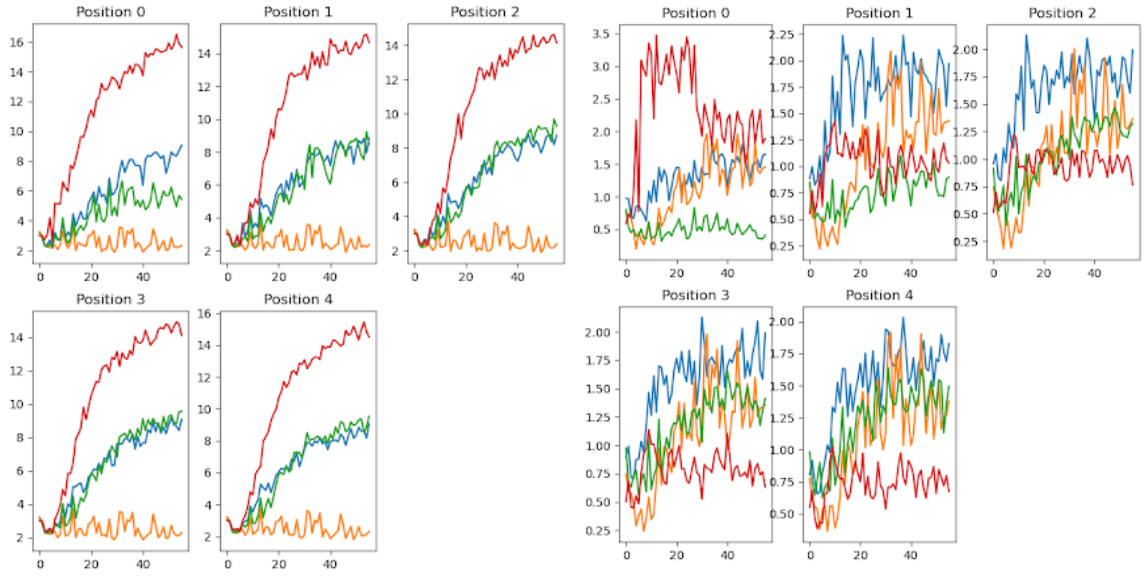


Figure 5.8. *Evolution of the mean compatibility values assigned to each head across a batch of samples for each of the first five positions of the input sentence during training. The values for the heads of the fifth layer are shown [5].*

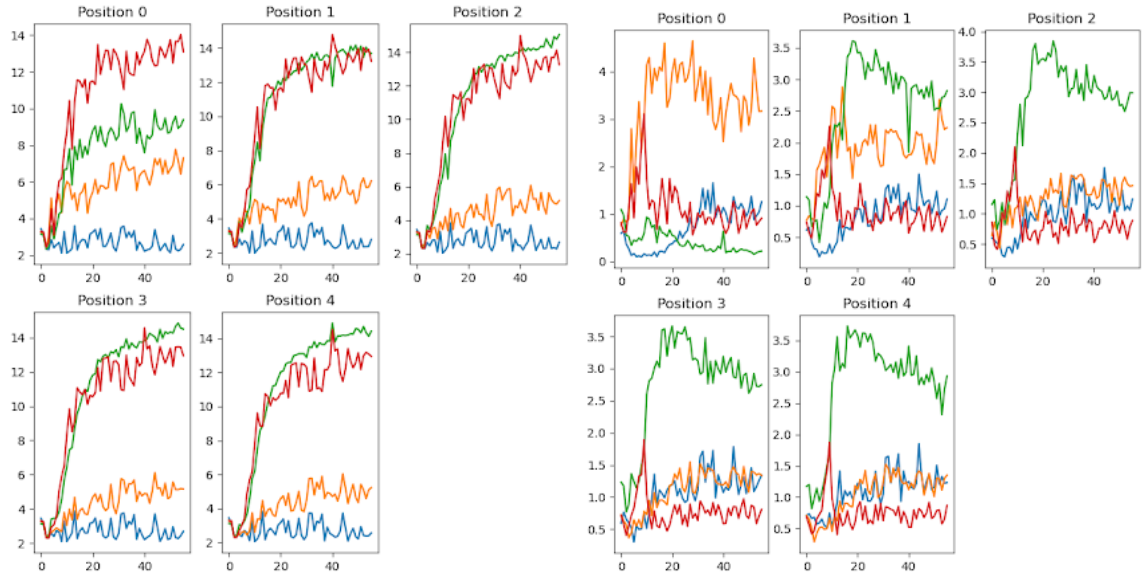
Using Attention to Promote Competition

Evidently, not all signals inside the Transformer model have norms close to each other. But, as they are modulated through their interaction with successive transformer layers, operations between them become feasible.



(a) *Before Multiplication - Third Layer* **(b)** *After Multiplication - Third Layer*

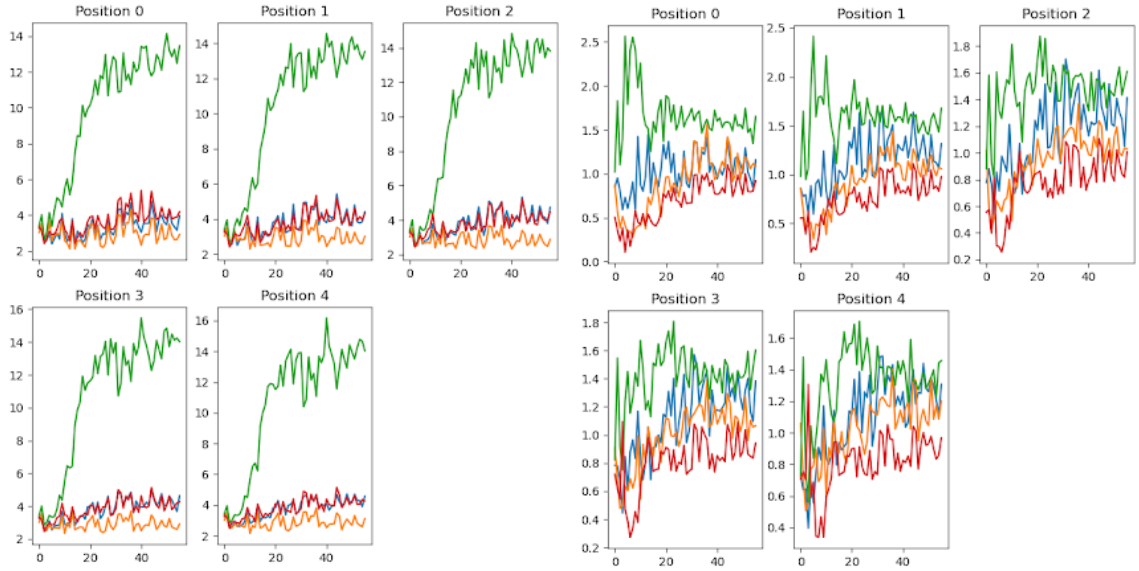
Figure 5.9. *Evolution of the mean norms of the heads' outputs before and after they are multiplied with the corresponding compatibility values across a batch of samples for each of the first five positions of the input sentence during training. The norms for the heads of the third layer are shown [5].*



(a) *Before Multiplication - Fourth Layer* **(b)** *After Multiplication - Fourth Layer*

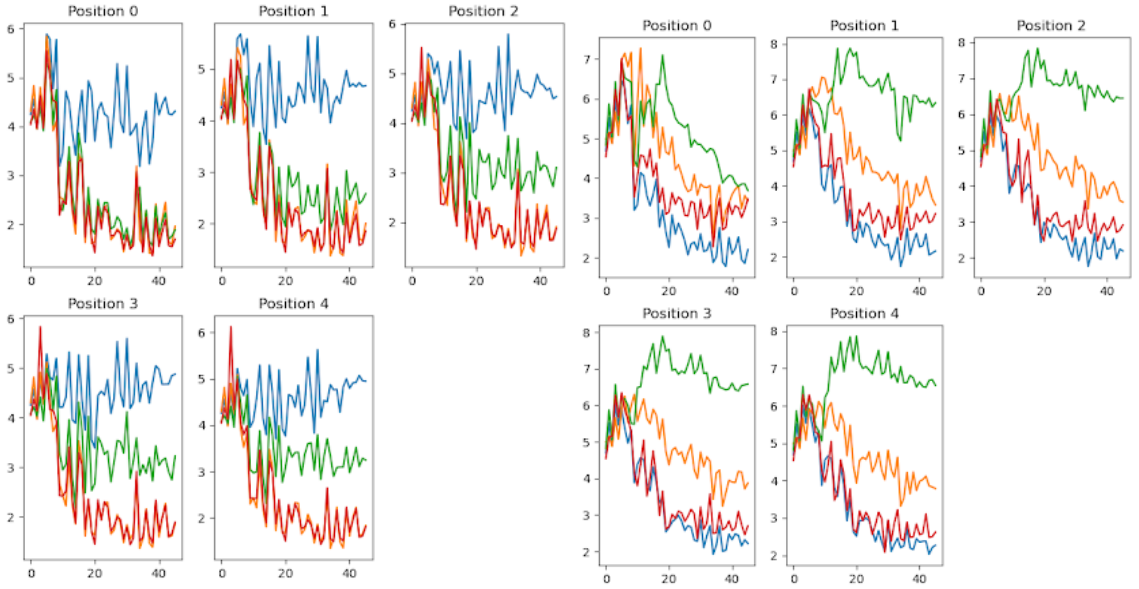
Figure 5.10. *Evolution of the mean norms of the heads' outputs before and after they are multiplied with the corresponding compatibility values across a batch of samples for each of the first five positions of the input sentence during training. The norms for the heads of the fourth layer are shown [5].*

This internal mechanism may provide the signals with a way to bypass the modulation applied by the attention mechanism implementing the notion of competition. If this is indeed the case, the model is able to adjust the norms of its signals according to the respective softmax scores, essentially cancelling our modification. A solution to this problem could be setting an attention threshold that would determine whether a head should be active or not, allowing only active attention heads to propagate their processing signals forward. This resembles the method employed by Goyal et al.



(a) *Before Multiplication - Fifth Layer* **(b)** *After Multiplication - Fifth Layer*

Figure 5.11. *Evolution of the mean norms of the heads' outputs before and after they are multiplied with the corresponding compatibility values across a batch of samples for each of the first five positions of the input sentence during training. The norms for the heads of the fifth layer are shown [5].*



(a) *Third Layer* **(b)** *Fourth Layer*

Figure 5.12. *Evolution of the mean norm values of the heads' outputs of the encoder's self-attention mechanism for a vanilla transformer model during its training. The values for the heads of the third and fourth layer of the model are shown [5].*

(2019) [2] to choose which RIMs should be active at each step. We could then choose to either set the output of the inactive attention heads to be the null vector, denoting lack of information, or the value vector of the corresponding input element, which is closer to the choice made by Goyal et al. (2019) for the inactive RIMs.

It must be noted that we implemented the method described above, setting the output of the inactive heads to be the null vector, but it actually lead to a deterioration of the model's

performance (table 5.3).

In any case, we claim that competition can prove to be a useful inductive bias but, as is true with many other inductive biases presented in this thesis, successfully implementing it is challenging and might require a less obvious modification than simply modulating the activation of neural experts.

But if we can get that...

We also plot the contribution metrics for the cases of six handpicked words, i.e. “but”, “if”, "we", “can”, “get” and “that” for the first and fourth layer (figures 5.14, 5.15, 5.16). Again, the contribution of each head, as computed by the matching mechanism, appears to be in general independent of the actual word being considered.

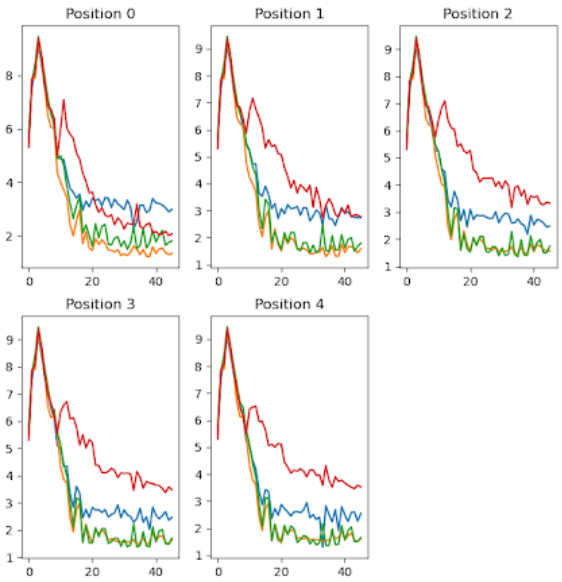


Figure 5.13. *Evolution of the mean norm values of the heads' outputs of the encoder's self-attention mechanism for a vanilla transformer model during its training. The values for the heads of the sixth layer of the model are shown [5].*

5.4.3 Application to A BERT-Based Model

A Robustly Optimized BERT Approach

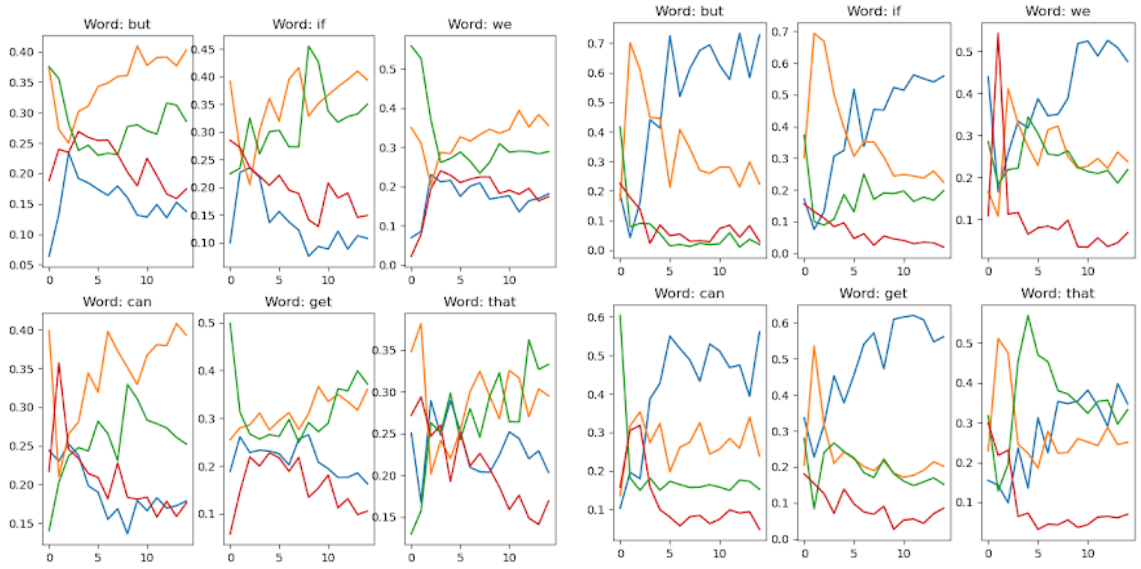
Liu et al. (2019) [52] perform modifications to the BERT's hyper-parameters and training process. They prove that a carefully tuned and trained BERT performs better than most of the BERT-based models that were publish in between. Specifically, they train the model for a longer period of time, with bigger batches, and using a bigger dataset, which they gathered, and on longer sentences. They also avoid using the next sentence prediction (NSP) objective and perform dynamic, instead of static, masking during training. After integrating all changes to the model and the training method, they train **RoBERTa**, a robustly optimized BERT approach. Their model outperforms sota models that were published in the meantime, after the publication of the BERT model and before theirs, on the GLUE dataset.

CAH in RoBERTa

They implement their model with FAIRSEQ, and we thus apply our method to it. We train the model on a masked language modeling (MLM) task using the English Wikipedia corpus. Due to time constraints we train the vanilla RoBERTa model for seven epochs and the RoBERTa with the CAH method applied to all of its attention mechanisms for eight epochs. After the seventh epoch of training for both models we report the PPL metric on the training set, which is equal to 12.55 for the vanilla model and 12.76 for the CAH version. We also report PPL on the validation set after the seventh epoch of training of the two models, which is equal to 10.04 for the standard RoBERTa model and 10.15 for the CAH version.

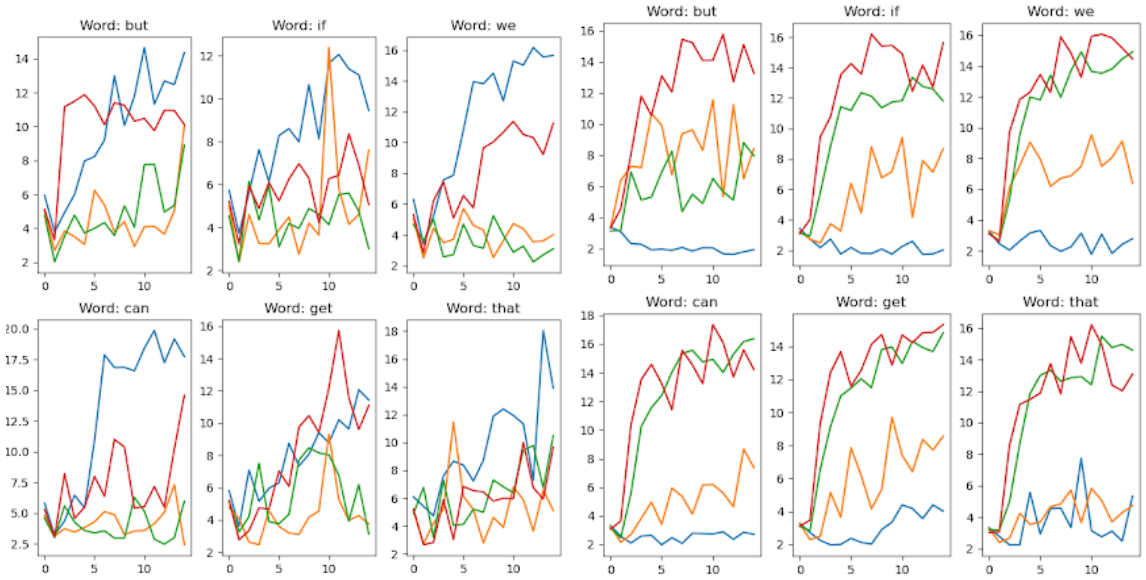
Training it in an iterative manner, the same way we did for the transformer model, also resulted in inferior performance compared to the vanilla model.

Like we did in the case of the transformer model, we include figures showing the evolution of the norms of the heads' outputs before and after the scalar-vector multiplication with the contribution



(a) *First Layer* **(b)** *Fourth Layer*

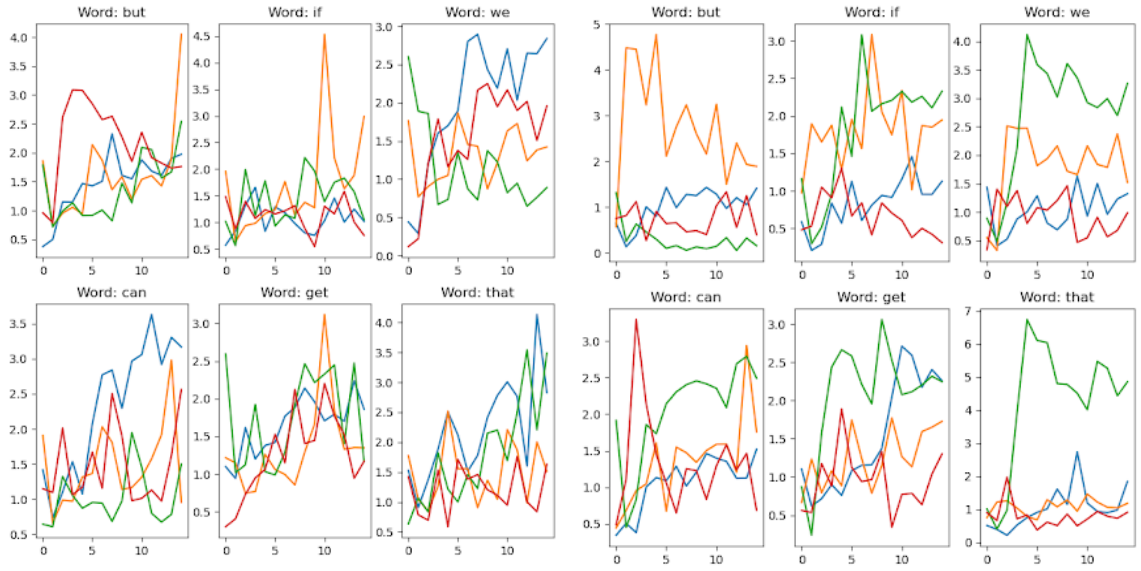
Figure 5.14. *Evolution of the mean compatibility values assigned to each head across all positions where the corresponding word is used as input during training. The values for the heads of the first and the fourth layers are shown [5].*



(a) *First Layer* **(b)** *Fourth Layer*

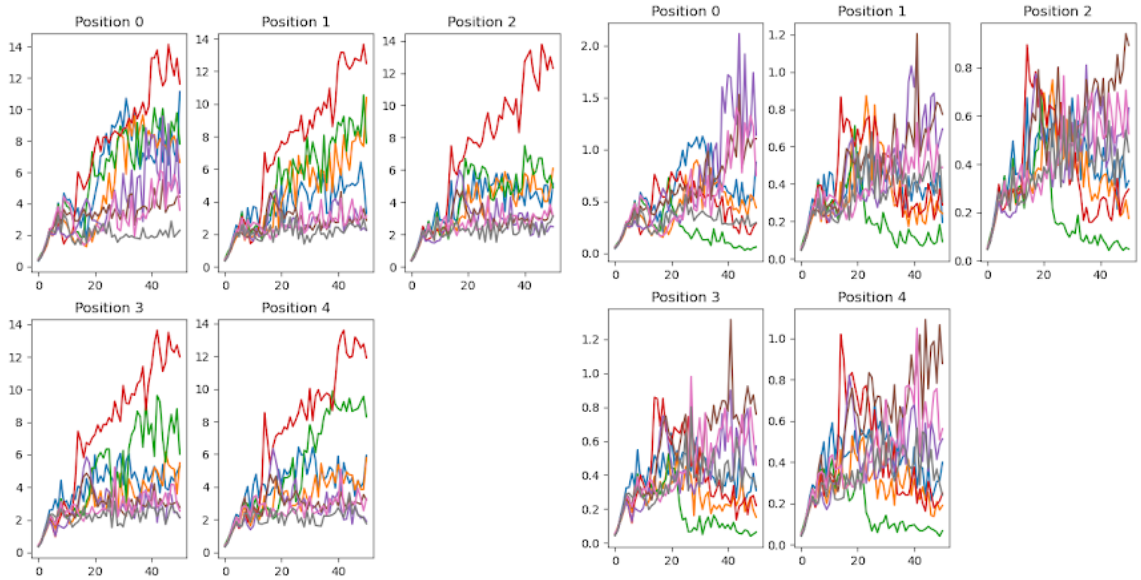
Figure 5.15. *Evolution of the mean norm values of the heads' outputs across all positions where the corresponding word is used as input during training before multiplication with the compatibility values is applied. The values for the heads of the first and fourth layer of the model are shown [5].*

values, as well as the contribution values themselves during training per position for the first five positions (figures 5.17, 5.18, 5.19, 5.20, 5.21). Again, no big differences between positions are evident. Moreover, we observe the same cancelling effect between the norms of the outputs and the respective compatibility values.



(a) *First Layer* **(b)** *Fourth Layer*

Figure 5.16. *Evolution of the mean norm values of the heads' outputs across all positions where the corresponding word is used as input during training after multiplication with the compatibility values is applied. The values for the heads of the first and fourth layer of the model are shown [5].*

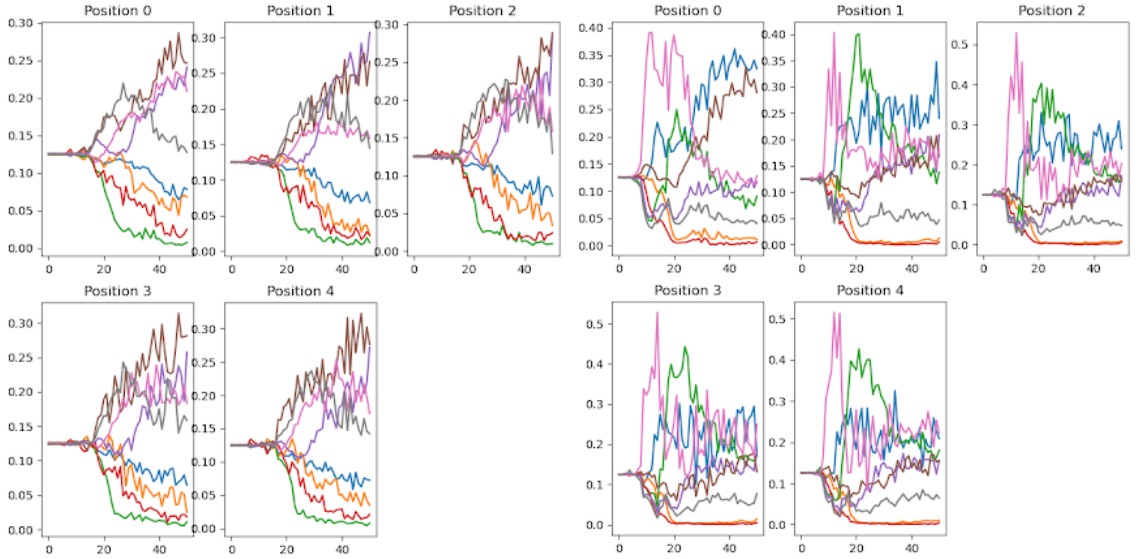


(a) *Before Multiplication - First Layer* **(b)** *After Multiplication - First Layer*

Figure 5.17. *Evolution of the mean norms of the heads' outputs before and after they are multiplied with the corresponding compatibility values across a batch of samples for each of the first five positions of the input sentence during training. The norms for the heads of the first layer are shown [5].*

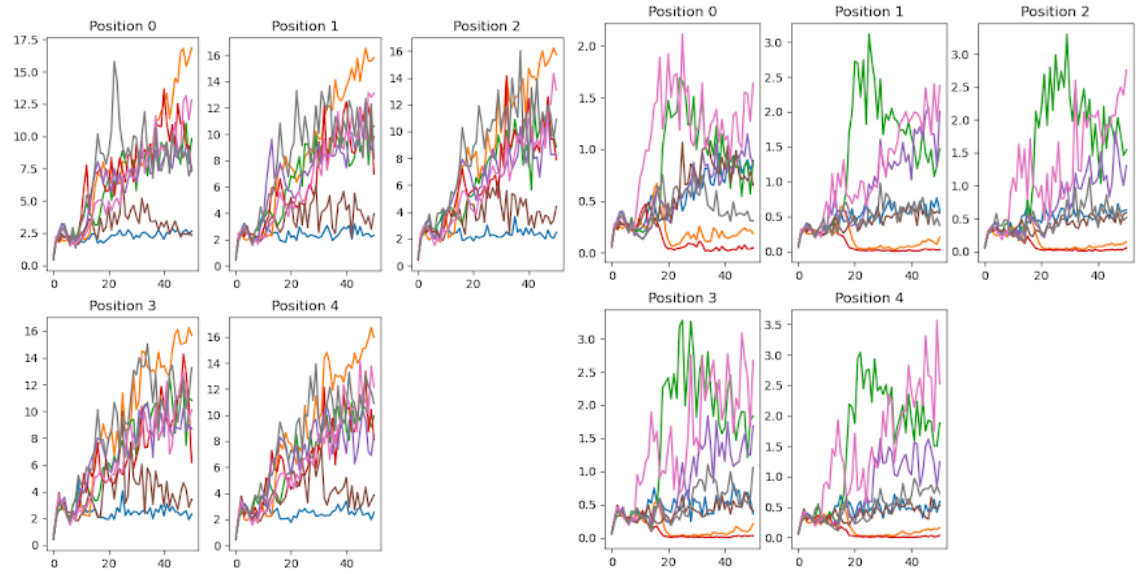
5.5 Discussion

The CAH mechanism did not overall improve the performance of the models it was applied to. We suspect that the model overfitted the mechanism. Out effort to train the mechanism and the model separately actually lead to a slight degradation in performance. We discussed how the solution may be found in a RIM-like activation of the relevant modules and a complete deactivation



(a) *First Layer* **(b)** *Fourth Layer*

Figure 5.18. *Evolution of the mean compatibility values assigned to each head across a batch of samples for each of the first five positions of the input sentence during training. The values for the heads of the first* ***(a)*** *and the fourth* ***(b)*** *layer are shown [5].*

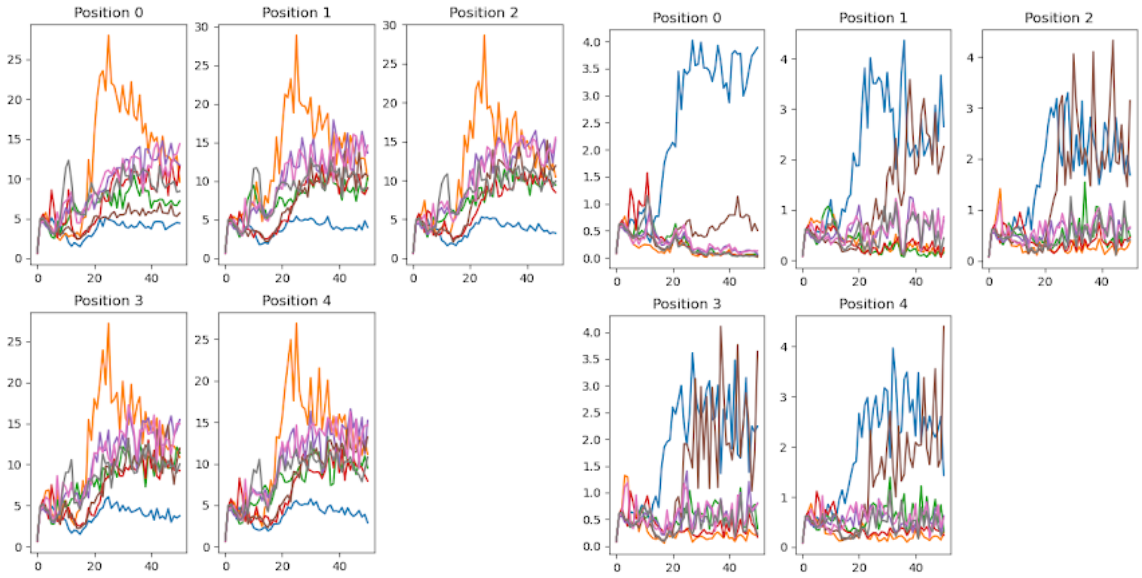


(a) *Before Multiplication - Fourth Layer* **(b)** *After Multiplication - Fourth Layer*

Figure 5.19. *Evolution of the mean norms of the heads' outputs before and after they are multiplied with the corresponding compatibility values across a batch of samples for each of the first five positions of the input sentence during training. The norms for the heads of the fourth layer are shown [5].*

of the irrelevant ones. Nevertheless, our preliminary experiments in that direction resulted in a drop in performance.

We still think that the use of a module that treats attention heads as mechanisms and promotes their specialization can improve the capabilities of attention-based models, and claim that even small changes in the way this module is implemented could unlock its potential. One could, for example, experiment with a variety of different decision metrics based on which head selection is performed, instead of using the signature vector approach. One could also opt for a more drastic



(a) *Before Multiplication - Eighth Layer* **(b)** *After Multiplication - Eighth Layer*

Figure 5.20. *Evolution of the mean norms of the heads' outputs before and after they are multiplied with the corresponding compatibility values across a batch of samples for each of the first five positions of the input sentence during training. The norms for the heads of the eighth layer are shown [5].*

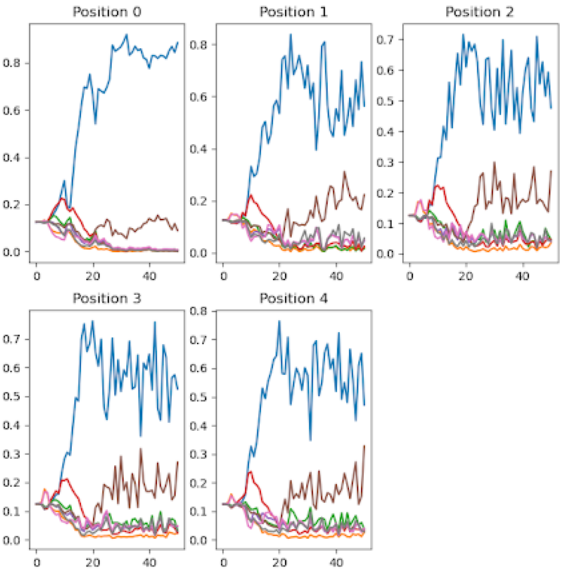


Figure 5.21. *Evolution of the mean compatibility values assigned to each head across a batch of samples for each of the first five positions of the input sentence during training. The values for the heads of the eighth layer are shown [5].*

modification to the module and even forgo the competitive character of the process altogether and employ a different method of promoting the specialization of heads, like an appropriate regularizer.

Chapter 6

Conclusions

6.1 Discussion

Concluding this thesis we believe that we have conducted a thorough review of the recent advances in efforts that aim to develop a generation of neural networks capable of performing higher cognitive functions. We explained how studies in the fields of cognitive science, neuroscience and causality support the intuition of designing neural architectures and training algorithms based on the set of inductive biases presented by Goyal and Bengio (2022) [51] (chapter 4.9). Given the important contribution of inductive biases that have already been implemented to the recent success of ML models we think that this is a direction which is certainly worthy of further investigation.

The models presented in the last sections of chapter 4 (4.10, 4.11, 4.12, 4.13) are inspired by the biases of independent mechanisms (chapter 4.6.5), specialization (chapter 4.3) and competition (chapter 4.6.7). Even though the reported results do not greatly surpass those of currently used methods and models, they pave the way for alternative implementations which may succeed in achieving the generalization abilities promised.

The CAH modification we proposed and applied to the transformer and BERT architectures did not improve their performance in general. Successfully implementing inductive biases has been proven to be a very hard task, as the research efforts we presented show. Yet, we certainly think that the notions of specialization and independent mechanisms will play important roles in the creation of the next generation of machine learning methods and models.

6.2 Future Work

There are many possible next steps one can take from here. An indicative list of such steps is provided below:

- OOD-generalization: As the biggest goal of the presented research efforts is to achieve good OOD generalization performance we consider testing our model under such conditions a natural next step.

- Meta-Learning: We tried training the inference mechanisms separately from the rest of the model, but our results were not satisfactory. Separating the model's parameters and training them in a meta-learning fashion (chapter 4.8) could prove to be useful.

- In the same spirit we consider notions that modify the training process, such as curriculum learning, a promising way to train our module.

- As we reported, we tried fully activating attention heads that are considered to be relevant in a given situation and fully deactivating irrelevant ones, but preliminary tests showed no

improvement. Nonetheless, we think that a more careful tuning of the hyper-parameters, such as the number of active heads could perhaps unlock the potential of the CAH method.

- Testing different metrics of relatedness, apart from the key - signature vector inner product, such as the attention weight statistics, is another promising direction.

Bibliography

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser και Illia Polosukhin. *Attention is all you need. Advances in neural information processing systems*, 30, 2017.

[2] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio και Bernhard Schölkopf. *Recurrent independent mechanisms. arXiv preprint arXiv:1909.10893*, 2019.

[3] Richard O Duda, Peter E Hart και David G Stork. *Pattern classification and scene analysis 2nd ed. ed: Wiley Interscience*, 13:14, 1995.

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

[5] J. D. Hunter. *Matplotlib: A 2D graphics environment. Computing in Science & Engineering*, 9(3):90–95, 2007.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren και Jian Sun. *Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition*, σελίδες 770–778, 2016.

[7] Ian Goodfellow, Yoshua Bengio και Aaron Courville. *Deep learning*. MIT press, 2016.

[8] Simon Haykin. *Neural networks and learning machines, 3/E*. Pearson Education India, 2009.

[9] Daniel Jurafsky και James H Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*.

[10] Dzmitry Bahdanau, Kyunghyun Cho και Yoshua Bengio. *Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473*, 2014.

[11] Jacob Devlin, Ming Wei Chang, Kenton Lee και Kristina Toutanova. *Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805*, 2018.

[12] Harold Hotelling. *Analysis of a complex of statistical variables into principal components. Journal of educational psychology*, 24(6):417, 1933.

[13] Jean Baptiste Cordonnier, Andreas Loukas και Martin Jaggi. *Multi-head attention: Collaborate instead of concatenate. arXiv preprint arXiv:2006.16362*, 2020.

[14] Wangchunshu Zhou, Tao Ge, Ke Xu, Furu Wei και Ming Zhou. *Scheduled drophead: A regularization method for transformer models. arXiv preprint arXiv:2004.13342*, 2020.

[15] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva και Antonio Torralba. *Object detectors emerge in deep scene cnns. arXiv preprint arXiv:1412.6856*, 2014.

[16] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich και Ivan Titov. *Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. arXiv preprint arXiv:1905.09418*, 2019.

[17] Kevin Clark, Urvashi Khandelwal, Omer Levy και Christopher D Manning. *What does bert look at? an analysis of bert's attention. arXiv preprint arXiv:1906.04341*, 2019.

[18] Joseph B Kruskal. *Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika*, 29(1):1–27, 1964.

[19] Jonas Peters, Dominik Janzing και Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms.* The MIT Press, 2017.

[20] Giambattista Parascandolo, Niki Kilbertus, Mateo Rojas-Carulla και Bernhard Schölkopf. *Learning independent causal mechanisms. International Conference on Machine Learning*, σελίδες 4036–4044. PMLR, 2018.

[21] Chelsea Finn και Sergey Levine. *Learning to learn: An Introduction to Meta Learning*, 2020.

[22] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell και others. *Language models are few-shot learners. Advances in neural information processing systems*, 33:1877–1901, 2020.

[23] Chelsea Finn, Pieter Abbeel και Sergey Levine. *Model-agnostic meta-learning for fast adaptation of deep networks. International conference on machine learning*, σελίδες 1126–1135. PMLR, 2017.

[24] Shawn Beaulieu, Lapo Frati, Thomas Miconi, Joel Lehman, Kenneth O Stanley, Jeff Clune και Nick Cheney. *Learning to continually learn. arXiv preprint arXiv:2002.09571*, 2020.

[25] Yoshua Bengio, Tristan Deleu, Nasim Rahaman, Rosemary Ke, Sébastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal και Christopher Pal. *A meta-transfer objective for learning to disentangle causal mechanisms. arXiv preprint arXiv:1901.10912*, 2019.

[26] Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu και Timothy Lillicrap. *Relational recurrent neural networks. Advances in neural information processing systems*, 31, 2018.

[27] Nasim Rahaman, Muhammad Waleed Gondal, Shruti Joshi, Peter Gehler, Yoshua Bengio, Francesco Locatello και Bernhard Schölkopf. *Dynamic inference with neural interpreters. Advances in Neural Information Processing Systems*, 34:10985–10998, 2021.

[28] Alex Lamb, Di He, Anirudh Goyal, Guolin Ke, Chien Feng Liao, Mirco Ravanelli και Yoshua Bengio. *Transformers with competitive ensembles of independent mechanisms. arXiv preprint arXiv:2103.00336*, 2021.

[29] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong και Richard Socher. *A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. arXiv preprint arXiv:1810.13243*, 2018.

[30] Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand και others. *Findings of the 2014 workshop on statistical machine translation. Proceedings of the ninth workshop on statistical machine translation*, σελίδες 12–58, 2014.

[31] Karim Ahmed, Nitish Shirish Keskar και Richard Socher. *Weighted transformer network for machine translation. arXiv preprint arXiv:1711.02132*, 2017.

[32] Yingce Xia, Tianyu He, Xu Tan, Fei Tian, Di He και Tao Qin. *Tied transformers: Neural machine translation with shared encoder and decoder. Proceedings of the AAAI conference on artificial intelligence*, τόμος 33, σελίδες 5466–5473, 2019.

[33] Tianyu He, Xu Tan, Yingce Xia, Di He, Tao Qin, Zhibo Chen και Tie Yan Liu. *Layer-wise coordination between encoder and decoder for neural machine translation. Advances in Neural Information Processing Systems*, 31, 2018.

[34] Alex Graves, Greg Wayne και Ivo Danihelka. *Neural turing machines. arXiv preprint arXiv:1410.5401*, 2014.

[35] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes και Yann LeCun. *Tracking the world state with recurrent entity networks. arXiv preprint arXiv:1612.03969*, 2016.

[36] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou και others. *Hybrid computing using a neural network with dynamic external memory. Nature*, 538(7626):471–476, 2016.

[37] David Barrett, Felix Hill, Adam Santoro, Ari Morcos και Timothy Lillicrap. *Measuring abstract reasoning in neural networks. International conference on machine learning*, σελίδες 511–520. PMLR, 2018.

[38] Xander Steenbrugge, Sam Leroux, Tim Verbelen και Bart Dhoedt. *Improving generalization for abstract reasoning tasks using disentangled feature representations. arXiv preprint arXiv:1811.04784*, 2018.

[39] Duo Wang, Mateja Jamnik και Pietro Lio. *Abstract diagrammatic reasoning with multiplex graph networks. arXiv preprint arXiv:2006.11197*, 2020.

[40] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly και others. *An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929*, 2020.

[41] Hyeong Seok Choi, Hoon Heo, Jie Hwan Lee και Kyogu Lee. *Phase-aware single-stage speech denoising and dereverberation with U-net. arXiv preprint arXiv:2006.00687*, 2020.

[42] Yuichiro Koyama, Tyler Vuong, Stefan Uhlich και Bhiksha Raj. *Exploring the best loss function for DNN-based low-latency speech enhancement with temporal convolutional networks. arXiv preprint arXiv:2005.11611*, 2020.

[43] Umut Isik, Ritwik Giri, Neerad Phansalkar, Jean Marc Valin, Karim Helwani και Arvindh Krishnaswamy. *Poconet: Better speech enhancement with frequency-positional embeddings, semi-supervised conversational data, and biased loss. arXiv preprint arXiv:2008.04470*, 2020.

[44] Bernard J Baars. *A cognitive theory of consciousness*. Cambridge University Press, 1993.

[45] Bernard J Baars. *In the theatre of consciousness. Global workspace theory, a rigorous scientific theory of consciousness. Journal of consciousness Studies*, 4(4):292–309, 1997.

[46] Bernard J Baars. *The global brainweb: An update on global workspace theory. Science and Consciousness Review*, 2, 2003.

[47] Bernard J Baars, Stan Franklin και Thomas Zoega Ramsoy. *Global workspace dynamics: cortical "binding and propagation" enables conscious contents. Frontiers in psychology*, 4:200, 2013.

[48] BJ Baars και N Geld. *On Consciousness: Science & Subjectivity-Updated Works on Global Workspace Theory*, 2019.

[49] Jimmy Lei Ba, Jamie Ryan Kiros και Geoffrey E Hinton. *Layer normalization. arXiv preprint arXiv:1607.06450*, 2016.

[50] Kahneman Daniel. *Thinking, fast and slow.* 2017.

[51] Anirudh Goyal και Yoshua Bengio. *Inductive biases for deep learning of higher-level cognition. Proceedings of the Royal Society A*, 478(2266):20210068, 2022.

[52] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer και Veselin Stoyanov. *Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692*, 2019.

[53] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier και Michael Auli. *fairseq: A Fast, Extensible Toolkit for Sequence Modeling. Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[54] Nicolas Palanca-Castan, Beatriz Sánchez Tajadura και Rodrigo Cofré. *Towards an interdisciplinary framework about intelligence. Heliyon*, 7(2):e06268, 2021.

[55] Levi Gadye. *The Tools That Let Neuroscientists Study (and Even Repair) Brain Circuits*, 2018.

[56] Warren S McCulloch και Walter Pitts. *A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics*, 5:115–133, 1943.

[57] Alan M Turing. *Computing machinery and intelligence.* Springer, 2009.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren και Jian Sun. *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Proceedings of the IEEE international conference on computer vision*, σελίδες 1026–1034, 2015.

[59] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot και others. *Mastering the game of Go with deep neural networks and tree search. nature*, 529(7587):484–489, 2016.

[60] Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincentde Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony και others. *Mastering the game of Stratego with model-free multiagent reinforcement learning. Science*, 378(6623):990–996, 2022.

[61] Russell Stuart J και Peter Norvig. *Artificial Intelligence: A Modern Approach, 4th US ed. Retrieved October*, 29, 2021.

[62] Jake Frankenfield. *Natural Language Processing (NLP)*, 2021.

[63] Beverly Park Woolf. *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning.* Morgan Kaufmann, 2010.

[64] Abdulhamit Subasi. *Practical machine learning for data analysis using python*. Academic Press, 2020.

[65] Hossein Ashtari. *What Is Computer Vision? Meaning, Examples, and Applications in 2022*, 2021.

[66] John J. Craig. *Introduction to Robotics: Mechanics and Control (3rd Edition)*. Pearson, 2004.

[67] Sergios Theodoridis και Konstantinos Koutroumbas. *Pattern recognition*. Elsevier, 2006.

[68] Yassine Ouali, Céline Hudelot και Myriam Tami. *An overview of deep semi-supervised learning. arXiv preprint arXiv:2006.05278*, 2020.

[69] Richard S Sutton και Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[70] Andrew Ng. *CS229 Lecture notes. CS229 Lecture notes*, 1(1):1–3, 2000.

[71] *How the Brain Works*. DK, 2020.

[72] Frank Rosenblatt. *The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review*, 65(6):386, 1958.

[73] Patricia Smith Churchland και Terrence Joseph Sejnowski. *The computational brain*. MIT press, 1992.

[74] Kurt Hornik, Maxwell Stinchcombe και Halbert White. *Multilayer feedforward networks are universal approximators. Neural networks*, 2(5):359–366, 1989.

[75] George Cybenko. *Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[76] Razvan Pascanu, Tomas Mikolov και Yoshua Bengio. *On the difficulty of training recurrent neural networks. International conference on machine learning*, σελίδες 1310–1318. Pmlr, 2013.

[77] Giuseppe Pio Cannata. *Vanishing gradient in Deep Neural Networks*, 2021.

[78] Vinod Nair και Geoffrey E Hinton. *Rectified linear units improve restricted boltzmann machines. Proceedings of the 27th international conference on machine learning (ICML-10)*, σελίδες 807–814, 2010.

[79] Andrew L Maas, Awni Y Hannun, Andrew Y Ng και others. *Rectifier nonlinearities improve neural network acoustic models. Proc. icml*, τόμος 30, σελίδα 3. Atlanta, Georgia, USA, 2013.

[80] Yann LeCun, Larry Jackel, Leon Bottou, A Brunot, Corinna Cortes, John Denker, Harris Drucker, Isabelle Guyon, UA Muller, Eduard Sackinger και others. *Comparison of learning algorithms for handwritten digit recognition. International conference on artificial neural networks*, τόμος 60, σελίδες 53–60. Perth, Australia, 1995.

[81] Siddharth Krishna Kumar. *On weight initialization in deep neural networks. arXiv preprint arXiv:1704.08863*, 2017.

[82] John Duchi, Elad Hazan και Yoram Singer. *Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research*, 12(7), 2011.

[83] Diederik P Kingma και Jimmy Ba. *Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980*, 2014.

[84] Tomaso Poggio και Federico Girosi. *Networks for approximation and learning. Proceedings of the IEEE*, 78(9):1481–1497, 1990.

[85] Bernard Widrow και Eugene Walach. *Adaptive signal processing for adaptive control. IFAC Proceedings Volumes*, 16(9):7–12, 1983.

[86] Andrew R Barron. *Neural net approximation. Proc. 7th Yale workshop on adaptive and learning systems*, τόμος 1, σελίδες 69–72, 1992.

[87] Ken Ichi Funahashi. *On the approximate realization of continuous mappings by neural networks. Neural networks*, 2(3):183–192, 1989.

[88] Daniel L Chester. *Why two hidden layers are better than one. Proc. IJCNN, Washington, DC*, τόμος 1, σελίδες 265–268, 1990.

[89] Babak Hassibi και David Stork. *Second order derivatives for network pruning: Optimal brain surgeon. Advances in neural information processing systems*, 5, 1992.

[90] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever και Ruslan Salakhutdinov. *Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research*, 15(1):1929–1958, 2014.

[91] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever και Ruslan R Salakhutdinov. *Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580*, 2012.

[92] Priya C Bala. *Build Better Deep Learning Models with Batch and Layer Normalization*, 2021.

[93] Suarav Singla. *Why is Batch Normalization useful in Deep Neural Networks?*, 2020.

[94] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez και Ion Stoica. *Tune: A Research Platform for Distributed Model Selection and Training. arXiv preprint arXiv:1807.05118*, 2018.

[95] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta και Masanori Koyama. *Optuna: A next-generation hyperparameter optimization framework. Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, σελίδες 2623–2631, 2019.

[96] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard και Lawrence D Jackel. *Backpropagation applied to handwritten zip code recognition. Neural computation*, 1(4):541–551, 1989.

[97] Yann LeCun, Léon Bottou, Yoshua Bengio και Patrick Haffner. *Gradient-based learning applied to document recognition. Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[98] David E Rumelhart, Geoffrey E Hinton και Ronald J Williams. *Learning internal representations by error propagation.* Τεχνική Αναφορά με αριθμό, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[99] Samy Bengio, Oriol Vinyals, Navdeep Jaitly και Noam Shazeer. *Scheduled sampling for sequence prediction with recurrent neural networks. Advances in neural information processing systems*, 28, 2015.

[100] Mike Schuster και Kuldip K Paliwal. *Bidirectional recurrent neural networks. IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[101] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk και Yoshua Bengio. *Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078*, 2014.

[102] Ilya Sutskever, Oriol Vinyals και Quoc V Le. *Sequence to sequence learning with neural networks. Advances in neural information processing systems*, 27, 2014.

[103] Yoshua Bengio, Paolo Frasconi και Patrice Simard. *The problem of learning long-term dependencies in recurrent networks. IEEE international conference on neural networks*, σελίδες 1183–1188. IEEE, 1993.

[104] Yoshua Bengio, Patrice Simard και Paolo Frasconi. *Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks*, 5(2):157–166, 1994.

[105] Sepp Hochreiter και Jürgen Schmidhuber. *Long short-term memory. Neural computation*, 9(8):1735–1780, 1997.

[106] Karin Verspoor και Kevin Cohen. *Natural Language Processing*, σελίδες 1495–1498. 2013.

[107] John E Hopcroft, Rajeev Motwani και Jeffrey D Ullman. *Introduction to automata theory, languages, and computation. Acm Sigact News*, 32(1):60–65, 2001.

[108] Chetna Khanna. *Byte-Pair Encoding: Subword-based tokenization algorithm*, 2021.

[109] Martin F Porter. *An algorithm for suffix stripping. Program*, 14(3):130–137, 1980.

[110] Martin Joos. *Description of language design. The Journal of the Acoustical Society of America*, 22(6):701–707, 1950.

[111] Zellig S Harris. *Distributional structure. Word*, 10(2-3):146–162, 1954.

[112] John Firth. *A synopsis of linguistic theory, 1930-1955. Studies in linguistic analysis*, σελίδες 10–32, 1957.

[113] Gerard Salton. *The SMART retrieval system—experiments in automatic document processing*. Prentice-Hall, Inc., 1971.

[114] Hans Peter Luhn. *A statistical approach to mechanized encoding and searching of literary information. IBM Journal of research and development*, 1(4):309–317, 1957.

[115] Karen Sparck Jones. *A statistical interpretation of term specificity and its application in retrieval. Journal of documentation*, 28(1):11–21, 1972.

[116] Tomas Mikolov, Kai Chen, Greg Corrado και Jeffrey Dean. *Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781*, 2013.

[117] Yoshua Bengio, Réjean Ducharme και Pascal Vincent. *A neural probabilistic language model. Advances in neural information processing systems*, 13, 2000.

[118] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu και Pavel Kuksa. *Natural language processing (almost) from scratch. Journal of machine learning research*, 12(ARTICLE):2493–2537, 2011.

[119] Piotr Bojanowski, Edouard Grave, Armand Joulin και Tomas Mikolov. *Enriching word vectors with subword information. Transactions of the association for computational linguistics*, 5:135–146, 2017.

[120] Jeffrey Pennington, Richard Socher και Christopher D Manning. *Glove: Global vectors for word representation. Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, σελίδες 1532–1543, 2014.

[121] Eric H Huang, Richard Socher, Christopher D Manning και Andrew Y Ng. *Improving word representations via global context and multiple word prototypes. Proceedings of the 50th annual meeting of the association for computational linguistics (Volume 1: Long papers)*, σελίδες 873–882, 2012.

[122] Mohammad Taher Pilehvar και Jose Camacho-Collados. *WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. arXiv preprint arXiv:1808.09121*, 2018.

[123] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman και Eytan Ruppin. *Placing search in context: The concept revisited. Proceedings of the 10th international conference on World Wide Web*, σελίδες 406–414, 2001.

[124] Felix Hill, Roi Reichart και Anna Korhonen. *Simlex-999: Evaluating semantic models with (genuine) similarity estimation. Computational Linguistics*, 41(4):665–695, 2015.

[125] Peter D Turney και Michael L Littman. *Corpus-based learning of analogies and semantic relations. Machine Learning*, 60:251–278, 2005.

[126] Tomáš Mikolov, Wen tau Yih και Geoffrey Zweig. *Linguistic regularities in continuous space word representations. Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, σελίδες 746–751, 2013.

[127] Anna Gladkova, Aleksandr Drozd και Satoshi Matsuoka. *Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't. Proceedings of the NAACL Student Research Workshop*, σελίδες 8–15, 2016.

[128] George A Miller και JG Beebe-Center. *Some psychological methods for evaluating the quality of translations.* Τεχνική Αναφορά με αριθμό, HARVARD UNIV CAMBRIDGE MA PSYCHOACOUSTIC LAB, 1956.

[129] Maja Popović. *chrF: character n-gram F-score for automatic MT evaluation. Proceedings of the tenth workshop on statistical machine translation*, σελίδες 392–395, 2015.

[130] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger και Yoav Artzi. *Bertscore: Evaluating text generation with bert. arXiv preprint arXiv:1904.09675*, 2019.

[131] Ricardo Rei, Craig Stewart, Ana C Farinha και Alon Lavie. *COMET: A neural framework for MT evaluation. arXiv preprint arXiv:2009.09025*, 2020.

[132] Thibault Sellam, Dipanjan Das και Ankur P Parikh. *BLEURT: Learning robust metrics for text generation. arXiv preprint arXiv:2004.04696*, 2020.

[133] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma και Andrej Risteski. *Linear algebraic structure of word senses, with applications to polysemy. Transactions of the Association for Computational Linguistics*, 6:483–495, 2018.

[134] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever και others. *Improving language understanding by generative pre-training.* 2018.

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

[136] Wilson L Taylor. *"Cloze procedure": A new tool for measuring readability. Journalism quarterly*, 30(4):415–433, 1953.

[137] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba και Sanja Fidler. *Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. Proceedings of the IEEE international conference on computer vision*, σελίδες 19–27, 2015.

[138] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy και Samuel R Bowman. *GLUE: A multi-task benchmark and analysis platform for natural language understanding. arXiv preprint arXiv:1804.07461*, 2018.

[139] Alex Warstadt, Amanpreet Singh και Samuel R Bowman. *Neural network acceptability judgments. Transactions of the Association for Computational Linguistics*, 7:625–641, 2019.

[140] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng και Christopher Potts. *Recursive deep models for semantic compositionality over a sentiment treebank. Proceedings of the 2013 conference on empirical methods in natural language processing*, σελίδες 1631–1642, 2013.

[141] Bill Dolan και Chris Brockett. *Automatically constructing a corpus of sentential paraphrases. Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[142] Zihan Chen, Hongbo Zhang, Xiaoji Zhang και Leqi Zhao. *Quora question pairs*, 2018.

[143] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio και Lucia Specia. *Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. arXiv preprint arXiv:1708.00055*, 2017.

[144] Adina Williams, Nikita Nangia και Samuel R Bowman. *A broad-coverage challenge corpus for sentence understanding through inference. arXiv preprint arXiv:1704.05426*, 2017.

[145] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev και Percy Liang. *Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05250*, 2016.

[146] Luisa Bentivogli, Peter Clark, Ido Dagan και Danilo Giampiccolo. *The Fifth PASCAL Recognizing Textual Entailment Challenge. TAC.* Citeseer, 2009.

[147] Hector Levesque, Ernest Davis και Leora Morgenstern. *The winograd schema challenge. Thirteenth international conference on the principles of knowledge representation and reasoning*, 2012.

[148] Rowan Zellers, Yonatan Bisk, Roy Schwartz και Yejin Choi. *Swag: A large-scale adversarial dataset for grounded commonsense inference. arXiv preprint arXiv:1808.05326*, 2018.

[149] Paul Michel, Omer Levy και Graham Neubig. *Are sixteen heads really better than one? Advances in neural information processing systems*, 32, 2019.

[150] *OpenAI: Introducing ChatGPT*, 2022.

[151] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko και others. *Highly accurate protein structure prediction with AlphaFold. Nature*, 596(7873):583–589, 2021.

[152] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard και David McClosky. *The Stanford CoreNLP natural language processing toolkit. Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, σελίδες 55–60, 2014.

[153] Mukund Sundararajan, Ankur Taly και Qiqi Yan. *Axiomatic attribution for deep networks. International conference on machine learning*, σελίδες 3319–3328. PMLR, 2017.

[154] Bernard J Baars, Natalie Geld και Robert Kozma. *Global workspace theory (GWT) and prefrontal cortex: Recent developments. Frontiers in psychology*, σελίδα 5163, 2021.

[155] Vladimir N Vapnik. *An overview of statistical learning theory. IEEE transactions on neural networks*, 10(5):988–999, 1999.

[156] Jonas Martin Peters. *Restricted structural equation models for causal inference.* Διδακτορική Διατριβή, ETH Zurich, 2012.

[157] Povilas Daniusis, Dominik Janzing, Joris Mooij, Jakob Zscheischler, Bastian Steudel, Kun Zhang και Bernhard Schölkopf. *Inferring deterministic causal relations. arXiv preprint arXiv:1203.3475*, 2012.

[158] Bernhard Schölkopf, Dominik Janzing, Jonas Peters, Eleni Sgouritsa, Kun Zhang και Joris Mooij. *On causal and anticausal learning. arXiv preprint arXiv:1206.6471*, 2012.

[159] Yann LeCun, Corinna Cortes, Chris Burges και others. *MNIST handwritten digit database*, 2010.

[160] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville και Yoshua Bengio. *Generative adversarial networks. Communications of the ACM*, 63(11):139–144, 2020.

[161] Brenden Lake, Ruslan Salakhutdinov, Jason Gross και Joshua Tenenbaum. *One shot learning of simple visual concepts. Proceedings of the annual meeting of the cognitive science society*, τόμος 33, 2011.

[162] Rich Caruna. *Multitask learning: A knowledge-based source of inductive bias. Machine learning: Proceedings of the tenth international conference*, σελίδες 41–48, 1993.

[163] Lorien Y Pratt. *Discriminability-based transfer between neural networks. Advances in neural information processing systems*, 5, 1992.

[164] Sebastian Ruder. *Transfer Learning - Machine Learning's Next Frontier*, 2017.

[165] Rupesh Kumar Srivastava, Klaus Greff και Jürgen Schmidhuber. *Highway networks. arXiv preprint arXiv:1505.00387*, 2015.

[166] Yoshua Bengio. *Deep learning of representations: Looking forward. Statistical Language and Speech Processing: First International Conference, SLSP 2013, Tarragona, Spain, July 29-31, 2013. Proceedings 1*, σελίδες 1–37. Springer, 2013.

[167] Vihar Kurama. *An Introduction to Decision Trees*, 2020.

[168] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra και others. *Matching networks for one shot learning. Advances in neural information processing systems*, 29, 2016.

[169] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. Διδακτορική Διατριβή, Technische Universität München, 1987.

[170] Yoshua Bengio, Samy Bengio και Jocelyn Cloutier. *Learning a synaptic learning rule*. Citeseer, 1990.

[171] Jake Snell, Kevin Swersky και Richard Zemel. *Prototypical networks for few-shot learning. Advances in neural information processing systems*, 30, 2017.

[172] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy και Samuel Bowman. *Superglue: A stickier benchmark for general-purpose language understanding systems. Advances in neural information processing systems*, 32, 2019.

[173] Trapit Bansal, Rishikesh Jha και Andrew McCallum. *Learning to few-shot learn across diverse natural language classification tasks. arXiv preprint arXiv:1911.03863*, 2019.

[174] Mark B Ring. *CHILD: A first step towards continual learning. Machine Learning*, 28(1):77–104, 1997.

[175] Michael McCloskey και Neal J Cohen. *Catastrophic interference in connectionist networks: The sequential learning problem. Psychology of learning and motivation*, τόμος 24, σελίδες 109–165. Elsevier, 1989.

[176] Roger Ratcliff. *Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. Psychological review*, 97(2):285, 1990.

[177] Rahaf Aljundi, Marcus Rohrbach και Tinne Tuytelaars. *Selfless sequential learning. arXiv preprint arXiv:1806.05421*, 2018.

[178] Magdalena Biesialska, Katarzyna Biesialska και Marta R Costa-Jussa. *Continual lifelong learning in natural language processing: A survey. arXiv preprint arXiv:2012.09823*, 2020.

[179] James L McClelland, Bruce L McNaughton και Randall C O'Reilly. *Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. Psychological review*, 102(3):419, 1995.

[180] Zhizhong Li και Derek Hoiem. *Learning without forgetting. IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[181] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska και others. *Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[182] Geoffrey Hinton, Oriol Vinyals και Jeff Dean. *Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531*, 2015.

[183] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu και Raia Hadsell. *Progressive neural networks. arXiv preprint arXiv:1606.04671*, 2016.

[184] Jaehong Yoon, Eunho Yang, Jeongtae Lee και Sung Ju Hwang. *Lifelong learning with dynamically expandable networks. arXiv preprint arXiv:1708.01547*, 2017.

[185] Ju Xu και Zhanxing Zhu. *Reinforced continual learning. Advances in Neural Information Processing Systems*, 31, 2018.

[186] Khurram Javed και Martha White. *Meta-learning representations for continual learning. Advances in neural information processing systems*, 32, 2019.

[187] Brenden Lake και Marco Baroni. *Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. International conference on machine learning*, σελίδες 2873–2882. PMLR, 2018.

[188] Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harmde Vries και Aaron Courville. *Systematic generalization: What is required and can it be learned? arXiv preprint arXiv:1811.12889*, 2018.

[189] Yoshua Bengio, Grégoire Mesnil, Yann Dauphin και Salah Rifai. *Better mixing via deep representations. International conference on machine learning*, σελίδες 552–560. PMLR, 2013.

[190] Michael F Mathieu, Junbo Jake Zhao, Junbo Zhao, Aditya Ramesh, Pablo Sprechmann και Yann LeCun. *Disentangling factors of variation in deep representation using adversarial training. Advances in neural information processing systems*, 29, 2016.

[191] Robert Desimone και John Duncan. *Neural mechanisms of selective visual attention. Annual review of neuroscience*, 18(1):193–222, 1995.

[192] Louis Kirsch, Julius Kunze και David Barber. *Modular networks: Learning to decompose neural computation. Advances in neural information processing systems*, 31, 2018.

[193] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu και Andrew Y Ng. *Reading digits in natural images with unsupervised feature learning.* 2011.

[194] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand και Victor Lempitsky. *Domain-adversarial training of neural networks. The journal of machine learning research*, 17(1):2096–2030, 2016.

[195] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto και David Ha. *Deep learning for classical japanese literature. arXiv preprint arXiv:1812.01718*, 2018.

[196] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku και Dustin Tran. *Image transformer. International conference on machine learning*, σελίδες 4055–4064. PMLR, 2018.

[197] Antony W Rix, John G Beerends, Michael P Hollier και Andries P Hekstra. *Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs. 2001 IEEE international conference on acoustics, speech, and signal processing. Proceedings (Cat. No. 01CH37221)*, τόμος 2, σελίδες 749–752. IEEE, 2001.

[198] Joachim Thiemann, Nobutaka Ito και Emmanuel Vincent. *The diverse environments multichannel acoustic noise database (demand): A database of multichannel environmental noise recordings. Proceedings of Meetings on Acoustics ICA2013*, τόμος 19, σελίδα 035081. Acoustical Society of America, 2013.

[199] Rohit Girdhar και Deva Ramanan. *CATER: A diagnostic dataset for Compositional Actions and TEmporal Reasoning. arXiv preprint arXiv:1910.04744*, 2019.

[200] HW Krohne, Neil J Smelser και PB Baltes. *International encyclopedia of the social & behavioral sciences*, 2001.

[201] Pentti Kanerva. *Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. Cognitive computation*, 1:139–159, 2009.

[202] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai και Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32*, σελίδες 8024–8035. Curran Associates, Inc., 2019.

[203] Kishore Papineni, Salim Roukos, Todd Ward και Wei Jing Zhu. *Bleu: a method for automatic evaluation of machine translation. Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, σελίδες 311–318, 2002.

List of Abbreviations

AI	Artificial Intelligence
ANML	A Neuromodulated Meta-Learning algorithm
BERT	Bidirectional Encoder Representations from Transformers
BLEU	BiLingual Evaluation Understudy
BPE	Byte-Pair Encoding
CAH	Competitive Attention Heads
CATER	Compositional Actions and TEmporal Reasoning
CF	Catastrophic Forgetting
CE	Cross-Entropy error function
CIFAR	Canadian Institute For Advanced Research
CL	Continual Learning
CNN	Convolutional Neural Network
CoLA	Corpus of Linguistic Acceptability
DL	Deep Learning
DNS	Deep Noise Suppression
ELMo	Embeddings from Language Models
FFNN	Feed-Forward Neural Network
GAN	Generative Adversarial Network
GD	Gradient Descent
GLLs	Group Linear Layers
GLUE	General Language Understanding Evaluation
GPT	Generative Pre-Training
GWT	Global Workspace Theory
GWD	Global Workspace Dynamics
ICM	Independence of Cause and Mechanism
iid	independent and identically distributed
KMNIST	Kuzushiji-MNIST
LM	Language Modeling
LEOPARD	(LEarning to generate sOftmax PARameters for Diverse classification
LOC	Line Of Code
LR	Linear Regression
LSA	Latent Semantic Analysis
LSTM	Long short-term memory
LRP	Layer-wise Relevance Propagation
MAML	Model-Agnostic Meta-Learning
MAP	Maximum A Posteriori estimation
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLM	Masked Language Modeling
MLP	Multilayer Perceptron

MNIST	Modified National Institute of Standards and Technology
MNLI	The Multi-Genre Natural Language Inference
ModAttn	Modulated Attention
ModLin	Modulated Linear Layer
MRPC	The Microsoft Research Paraphrase Corpus
MSE	Mean Square Error
NER	Named Entity Recognition
NIs	Neural Interpreters
NLI	Natural Language Inference
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
NLM	Neural Language Modeling
NMT	Neural Machine Translation
NN	Neural Network
NNLM	Neural Network Language Model
NSP	Next Sentence Prediction
OML	Online Aware Meta-Learning
OOD	Out-Of-Distribution
PGMs	Progressively Generated Matrices
PPL	PerPLexity
QA	Question Answering
QNLI	Question-answering Natural Language Inference
QQP	Quora Question Pairs
ReLU	Rectified Linear Unit
RIMs	Recurrent Independent Mechanisms
RNN	Recurrent Neural Network
RoBERTa	Robustly optimized BERT approach
RTE	Recognizing Textual Entailment
SCM	Structural Causal Model
SST-2	The Stanford Sentiment Treebank
STS-B	Semantic Textual Similarity Benchmark
SQuAD	Stanford Question Answering Dataset
SVHN	Street View House Numbers Dataset
TIMs	Transformers with competitive ensembles of Independent Mechanisms
WNLI	Winograd Natural Language Inference