ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Σημασιολογικός Εμπλουτισμός Προεκπαιδευμένων Νευρωνικών Δικτύων για την Επεξεργασία Φυσικής Γλώσσας με τη Χρήση Εργαλείων Αναπαράστασης Γνώσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασιλική Α. Ξεφτέρη

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών

Σημασιολογικός Εμπλουτισμός Προεκπαιδευμένων Νευρωνικών Δικτύων για την Επεξεργασία Φυσικής Γλώσσας με τη Χρήση Εργαλείων Αναπαράστασης Γνώσης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Βασιλική Α. Ξεφτέρη

Επιβλέπων: Γεώργιος Στάμου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 13η Ιουλίου 2022.

.....	.....	.....
Γεώργιος Στάμου	Αθανάσιος Βουλόδημος	Ανδρέας-Γεώργιος Σταφυλοπάτης
Καθηγητής Ε.Μ.Π.	Επίκουρος Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2022

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

DIVISION OF COMPUTER SCIENCE

Semantic Enrichment of Pre-trained Neural Networks for Natural Language Processing with the Use of Knowledge Representation Tools

DIPLOMA THESIS

Vasiliki A. Xefteri

Supervisor: Georgios Stamou
NTUA Professor

Athens, July 2022

.....

Βασιλική Α. Ξεφτέρη
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Ολοκληρώνοντας τις σπουδές μου στο ΕΜΠ, θα ήθελα να ευχαριστήσω ιδιαίτερα τον επιβλέποντα καθηγητή μου κ. Στάμου για την εμπιστοσύνη που έδειξε στις δυνατότητές μου και τη συνεργασία μας. Επίσης, θα ήθελα να ευχαριστήσω τον Γιώργο Φιλανδριανό για την άμεση και εποικοδομητική συνεργασία μας στην υλοποίηση της παρούσας διπλωματικής. Τέλος, ένα μεγάλο ευχαριστώ στον αδελφό μου, τους γονείς μου και σε όλους τους φίλους μου για τη στήριξή τους σε όλη τη διάρκεια των σπουδών μου.

Περίληψη

Τα τελευταία χρόνια, η επιστημονική βιβλιογραφία στον βιοϊατρικό και κλινικό τομέα αυξάνεται ολοένα και περισσότερο. Αυτή η ταχεία ανάπτυξη έχει περιπλέξει τον εντοπισμό πληροφοριών που ενδιαφέρουν τους ερευνητές. Επί του παρόντος, υπάρχουν διάφορα εποπτευόμενα μοντέλα για την ανάκτηση πληροφοριών και την ταξινόμηση ιατρικών εγγράφων, ωστόσο η ταχεία εμφάνιση νέων θεμάτων και ευρημάτων συχνά εμποδίζει την απόδοσή τους. Η διπλωματική αυτή ασχολείται με τη διερεύνηση του σημασιολογικού εμπλουτισμού αρχιτεκτονικών βαθιάς μάθησης transformer με σκοπό την εύρεση των σχετικών ιατρικών εγγράφων με ένα ερώτημα χρήστη και επίσης την ταξινόμηση αυτών των εγγράφων. Αξιολογούμε τις μεθόδους μας σε υποσύνολα δεδομένων OHSUMED. Και στα δύο προβλήματα, ο σημασιολογικός εμπλουτισμός επιτυγχάνεται με τη SNOMED CT, μια οντολογία κλινικής υγειονομικής περίθαλψης, και χρησιμοποιούμε κυρίως δύο διαφορετικές προσεγγίσεις.

Σε αυτή τη διπλωματική, αρχικά, διερευνούμε τον εμπλουτισμό των ενσωματώσεων κειμένων που βασίζονται σε μοντέλα transformer με ενσωματώσεις οντολογιών, που παράγονται από το OWL2Vec* [11]. Το OWL2Vec* είναι ένα πλαίσιο που διατηρεί τις λεξιλογικές πληροφορίες και τους λογικούς τελεστές μιας οντολογίας. Πειραματιζόμαστε με διαφορετικές παραμέτρους του OWL2Vec* και διαφορετικές προ-διεργασίες του συνόλου δεδομένων και της οντολογίας μας και αποδεικνύουμε ότι το OWL2Vec* δεν μπορεί επί του παρόντος να εφαρμοστεί σε μεγάλες οντολογίες, όπως η SNOMED CT. Αυτό το αποδεικνύουμε όχι μόνο στην ανάκτηση πληροφοριών αλλά και στην ταξινόμηση κειμένων. Στο μέλλον, η απόδοση του OWL2Vec* αναμένεται να βελτιωθεί με μεγαλύτερες οντολογίες.

Για την ταξινόμηση, δοκιμάζουμε επίσης μια άλλη μέθοδο που βασίζεται στο φιλτράρισμα. Ενισχύουμε κάθε κλάση και κάθε έγγραφο με έννοιες της SNOMED CT και, στη συνέχεια, επιβάλλουμε φίλτρα στη συνύπαρξη εννοιών μεταξύ τους. Η μέθοδος επιτυγχάνει καλή απόδοση στην ταξινόμηση. Εξετάζουμε αυτή την προσέγγιση συνδυάζοντάς την με τα μοντέλα BERT και βελτιώνουμε σημαντικά την απόδοσή τους. Ως αποτέλεσμα, αποδεικνύουμε ότι μια εξειδικευμένη οντολογία μπορεί να εφαρμοστεί με επιτυχία για την προσαρ-

μογή μοντέλων, που δεν εξειδικεύονται σε έναν τομέα, σε έναν νέο τομέα και να βελτιώσει την απόδοση των μοντέλων επεξεργασίας φυσικής γλώσσας.

Λέξεις κλειδιά

BERT, SNOMED CT, OWL2Vec*, Σημασιολογικός εμπλουτισμός, Ανάκτηση πληροφορίας, Ταξινόμηση, Επεξεργασία φυσικής γλώσσας

Abstract

In recent years, the scientific literature in the biomedical and clinical domain is more and more increasing. This rapid growth has complicated the identification of information of interest by researchers. Various supervised models currently exist for the information retrieval and the classification of medical documents, however the rapid emergence of new topics and findings often hinders their performance. The thesis is concerned with investigating the semantic enrichment of deep transformer architectures in order to find the related medical documents with a user query and also to classify these documents. We evaluate our methods on subsets of OHSUMED dataset. On both tasks the semantic enrichment is achieved with SNOMED CT, a clinical healthcare terminology, and we use mainly two different approaches.

In this thesis, firstly, we investigate the enrichment of transformer-based embeddings with owl embeddings, produced by OWL2Vec* [11]. OWL2Vec* is a framework that preserves the lexical information and the logical constructors of an ontology. We experiment with different settings of OWL2Vec* and different pre-processes of our dataset and ontology and we prove that OWL2Vec* is currently unable to be applied in large ontologies, like SNOMED CT. We prove this not only on information retrieval but also on text classification. In the future the performance of OWL2Vec* is expected to improve with larger vocabularies.

For the classification task we, also, try another method based on filtering. We enhance each class and each document with SNOMED CT concepts and then impose filters on concept co-occurrence between them. The method achieves good performance on classification. We examine this approach while combining it with BERT models and improve significantly their performance. As result, we prove that a specialized ontology, can successfully be applied to adapt out-of-domain models to a new domain and improve the performance of natural language processing models.

Keywords

BERT, SNOMED CT, OWL2Vec*, Semantic enrichment, Information retrieval, Classification, Natural language processing

Contents

List of Figures

List of Tables

Κεφάλαιο 1

Εκτεταμένη Ελληνική Περίληψη

Αυτό το κεφάλαιο περιλαμβάνει μία περιληπτική παρουσιάση των περιεχομένων αυτής της διπλωματικής εργασίας στα ελληνικά.

1.1 Εισαγωγή

Τα τελευταία χρόνια, η ανάγκη εφαρμογής μοντέλων μηχανικής μάθησης σε διάφορους τομείς αυξάνεται ολοένα και περισσότερο. Για περισσότερα από 50 χρόνια, οι επιστήμονες έχουν επικεντρωθεί στην ανάπτυξη της ικανότητας των ηλεκτρονικών υπολογιστών για την κατανόηση της ανθρώπινης γλώσσας, γραπτής και προφορικής, που αναφέρεται ως φυσική γλώσσα. Αυτό το πεδίο που συνδιάζει την Γλωσσολογία, την Επιστήμη Υπολογιστών και την Τεχνητή Νοημοσύνη ονομάζεται Επεξεργασία Φυσικής Γλώσσας (NLP). Οι εφαρμογές της επεξεργασίας φυσικής γλώσσας αφορούν συνήθως την ανάκτηση πληροφορίας, την ομαδοποίηση πληροφορίας, την μηχανική μετάφραση, την περίληψη κειμένων, την ανάλυση συναισθημάτων, την ταξινόμηση και την αυτοματοποιημένη αναγνώριση ομιλίας.

Σήμερα, τα βαθιά νευρωνικά δίκτυα, όπως τα μοντέλα BERT, είναι ευρέως διαδεδομένα και αποτελεσματικά στο πεδίο της επεξεργασίας φυσικής γλώσσας. Κατανοούμε, φυσικά, ότι η συμβολή τους ιδιαίτερα στον τομέα της ιατρικής είναι πολύ σημαντική αφού η επεξεργασία φυσικής γλώσσας βοηθάει στην ανάλυση σημειώσεων και κειμένων, που βρίσκονται σε ηλεκτρονική μορφή, και τα οποία διαφορετικά θα ήταν απρόσιτα για μελέτη. Ενώ οι μέθοδοι μηχανικής εκμάθησης όμως βελτιώνονται συνεχώς, τα ιατρικά θέματα αλλάζουν γρήγορα και το ίδιο αλλάζουν και οι σχετικοί πόροι (επιστημονικές δημοσιεύσεις, εκθέσεις, κλινικές δοκιμές). Έτσι, αν και τα τυπικά σύνολα δεδομένων παρέχουν

μια σταθερή βάση για την εκπαίδευση, τη βελτίωση και την αξιολόγηση νέων μεθόδων, δεν μπορούν να λάβουν υπόψη αναδυόμενα θέματα, νέες οντότητες και ορολογίες.

Ο στόχος της διπλωματικής εργασίας είναι ο σημασιολογικός εμπλουτισμός των προεκπαιδευμένων νευρωνικών δικτύων για την επεξεργασία φυσικής γλώσσας με τη χρήση εργαλείων αναπαράστασης γνώσης. Συγκεκριμένα εστιάζουμε σε διαφορετικούς τρόπους βελτίωσης των ενσωματώσεων (embeddings) των μοντέλων BERT με γνώση που εξάγεται από την SNOMED CT. Η SNOMED CT αποτελεί την πιο ολοκληρωμένη ορολογία κλινικής υγειονομικής περίθαλψης, που αποτελείται από πάνω από 350.000 έννοιες και καλύπτουν κλινικά ευρήματα, συμπτώματα, διαγνώσεις, διαδικασίες, οργανισμούς και άλλες αιτιολογίες, ουσίες, φαρμακευτικά προϊόντα, συσκευές και δείγματα. Τα μοντέλα BERT, τα οποία ενσωματώνουμε, είναι μια οικογένεια υψηλών επιδόσεων προεκπαιδευμένων γλωσσικών μοντέλων που παράγουν αποτελέσματα αιχμής σε ένα ευρύ φάσμα προβλημάτων επεξεργασίας της φυσικής γλώσσας (NLP). Η βασική τεχνική καινοτομία του BERT είναι η εφαρμογή της αμφίδρομης εκπαίδευσης των Transformers στη μοντελοποίηση γλώσσας. Με τη χρήση πολλαπλών μηχανισμών προσοχής (πολλαπλή προσοχή), το μοντέλο είναι σε θέση να συλλάβει ένα ευρύτερο φάσμα σχέσεων μεταξύ των λέξεων από αυτό που θα ήταν δυνατό με έναν μόνο μηχανισμό προσοχής. Επιπλέον, το BERT στοιβάζει πολλαπλά στρώματα προσοχής, καθένα από τα οποία λειτουργεί στην έξοδο του στρώματος που ήρθε πριν. Μέσω αυτής της επαναλαμβανόμενης σύνθεσης ενσωματώσεων λέξεων, το BERT είναι σε θέση να σχηματίσει πολύ πλούσιες αναπαραστάσεις καθώς φτάνει στα βαθύτερα στρώματα του μοντέλου.

Επικεντρωνόμαστε στην ανάκτηση πληροφοριών και στην ταξινόμηση κειμένου. Για την αξιολόγηση χρησιμοποιούμε υποσύνολα του συνόλου δεδομένων OHSUMED ώστε να βαθμολογήσουμε τις μεθόδους μας σε σύγκριση με διαφορετικά μοντέλα BERT. Αρχικά, και για τις δύο εργασίες, εμπλουτίζουμε τις ενσωματώσεις κειμένων των μοντέλων BERT με ενσωματώσεις οντολογιών της SNOMED CT. Οι ενσωματώσεις οντολογιών για κάθε έγγραφο παράγονται χρησιμοποιώντας το OWL2Vec*, το οποίο διατηρεί τις λεξιλογικές πληροφορίες και τους λογικούς τελεστές μιας οντολογίας. Πειραματιζόμαστε με διαφορετικές ρυθμίσεις του OWL2Vec* και διαφορετικές προ-διεργασίες του συνόλου δεδομένων και της οντολογίας μας και αποδεικνύουμε ότι το OWL2Vec* δεν μπορεί να εφαρμοστεί επί του παρόντος σε μεγάλα λεξιλόγια, όπως η SNOMED CT. Χρησιμοποιούμε, επίσης, αυτόν τον σημασιολογικό εμπλουτισμό των ενσωματώσεων στην ταξινόμηση των εγγράφων και επιβεβαιώνουμε και πάλι την υπόθεσή μας.

Δοκιμάζουμε, επίσης, στην ταξινόμηση μια μέθοδο φιλτραρίσματος. Δημιουργούμε ένα φίλτρο στα έγγραφα που βασίζεται στη συνύπαρξη μεταξύ των εννοιών που σχετίζονται με την κλάση και των όρων των εγγράφων. Για την

εύρεση των εννοιών που σχετίζονται με την κλάση εκμεταλλευόμαστε την ιεραρχία εννοιών της SNOMED CT. Πειραματιζόμαστε με διαφορετικά βάθη στη SNOMED CT και διαφορετικά κατώφλια στα έγγραφα. Αποδεικνύουμε ότι τα μοντέλα BERT σε συνδυασμό με το φιλτράρισμα επιτυγχάνουν καλύτερη απόδοση από τα αφιλτράριστα μοντέλα. Ως αποτέλεσμα, αποδεικνύουμε ότι μια εξειδικευμένη οντολογία μπορεί να εφαρμοστεί με επιτυχία για την προσαρμογή μοντέλων, που δεν εξειδικεύονται σε έναν τομέα, σε έναν νέο τομέα και να βελτιώσει την απόδοση των μοντέλων επεξεργασίας φυσικής γλώσσας.

1.2 Αναπαράσταση Φυσικής Γλώσσας

Προτού αναφερθούμε σε κάποια μοντέλα επεξεργασίας φυσικής γλώσσας, είναι σημαντικό να αναλύσουμε τον τρόπο με τον οποίο οι υπολογιστές επεξεργάζονται τη φυσική γλώσσα. Στην πραγματικότητα οι υπολογιστές αναπαριστούν την γλώσσα ως διάνυσμα.

Οι αναπαραστάσεις της γλώσσας που δημιουργούνται χρησιμοποιώντας νευρωνικά δίκτυα αναφέρονται συνήθως ως ενσωματώσεις (embeddings). Η σημασιολογική ενσωμάτωση αναφέρεται σε μια σειρά τεχνικών εκμάθησης αναπαράστασης (ή εκμάθησης χαρακτηριστικών) που κωδικοποιούν τη σημασιολογία των δεδομένων, όπως οι ακολουθίες και τα γραφήματα σε διανύσματα, έτσι ώστε να μπορούν να χρησιμοποιηθούν από εργασίες πρόβλεψης μηχανικής μάθησης και στατιστικής ανάλυσης. Αρχικά θα αναφερθούμε στα θεμέλια πίσω από την κατασκευή σημασιολογικών χώρων, ειδικά για την δημιουργία των ενσωματώσεων λέξεων.

1.2.1 Ενσωματώσεις Λέξεων

Οι σημασιολογικοί χώροι κατασκευάζονται αυτόματα αναλύοντας την συνύπαρξη των λέξεων σε μεγάλα κείμενα. Λέξεις που εμφανίζονται σε παρόμοια συμφραζόμενα τείνουν να έχουν παρόμοια σημασία. Οι ενσωματώσεις λέξεων είναι στην πραγματικότητα ένας ειδικός τύπος κατανεμημένης αναπαράστασης λέξεων. Κατασκευάζονται με νευρωνικά δίκτυα, κυρίως διαδεδομένων μετά το 2013. Δύο από τα πιο γνωστά μοντέλα ενσωματώσεων λέξεων είναι το Word2vec και το GloVe.

Το Word2vec αναπτύχθηκε αρχικά από μια ομάδα της Google το 2013. Μόλις εκπαιδευτεί, ένα τέτοιο μοντέλο μπορεί να ανιχνεύσει συνώνυμες λέξεις ή να προτείνει πρόσθετες λέξεις για μια μερική πρόταση. Το Word2vec αντιπροσωπεύει κάθε ξεχωριστή λέξη με ένα διάνυσμα, το οποίο επιλέγεται προσεκτικά έτσι ώστε μια απλή μαθηματική συνάρτηση να υποδεικνύει το επίπεδο σημασιολογικής ομοιότητας μεταξύ των λέξεων.

Το Word2vec βασίζεται σε ένα απλό αλλά αποτελεσματικό νευρωνικό δίκτυο που εκπαιδεύεται με στόχο τη μοντελοποίηση γλώσσας. Το Word2vec μπορεί να χρησιμοποιεί οποιαδήποτε από τις δύο κλασικές αρχιτεκτονικές αυτόματης κωδικοποίησης για εκμάθηση αναπαραστάσεων διαδοχικών στοιχείων: συνεχής Skip-gram και συνεχής Bag-of-Words (CBOW). Το μοντέλο CBOW στοχεύει στην πρόβλεψη της τρέχουσας λέξης χρησιμοποιώντας το περιβάλλον της. Το μοντέλο Skip-gram είναι παρόμοιο με το μοντέλο CBOW αλλά ο στόχος είναι να προβλέψει τις λέξεις στο περιβάλλον δεδομένης της λέξης-στόχος, αντί να προβλέπει την ίδια τη λέξη-στόχο.

Το GloVe αναπτύχθηκε στο Stanford το 2014. Το μοντέλο είναι ένας αλγόριθμος μάθησης χωρίς επίβλεψη για τη λήψη διανυσματικών αναπαραστάσεων για λέξεις. Αυτό επιτυγχάνεται με την αντιστοίχιση λέξεων σε ένα χώρο με νόημα όπου η απόσταση μεταξύ των λέξεων σχετίζεται με τη σημασιολογική ομοιότητα. Η εκπαίδευση εκτελείται σε συγκεντρωτικά καθολικά στατιστικά στοιχεία συνύπαρξης λέξεων από ένα κείμενο.

1.2.2 Ενσωματώσεις με βάση τα Συμφραζόμενα

Το πρόβλημα των προεκπαιδευμένων ενσωματώσεων λέξεων είναι ότι υπολογίζουν μια στατική αναπαράσταση για κάθε λέξη αφού η αναπαράσταση είναι ανεξάρτητη από το περιεχόμενο στο οποίο εμφανίζεται η λέξη. Αυτό μπορεί να εμποδίσει την ικανότητα των συστημάτων NLP να κατανοούν τη σημασιολογία ενός κειμένου. Αντίθετα οι ενσωματώσεις με βάση τα συμφραζόμενα είναι δυναμικές και η ίδια λέξη μπορεί να έχει διαφορετικές ενσωματώσεις εάν εμφανίζεται σε διαφορετικά πλαίσια. Τα μοντέλα αυτά λαμβάνουν ολόκληρο το κείμενο (τη λέξη-στόχο μαζί με το περιεχόμενό της) και παρέχουν εξειδικευμένες ενσωματώσεις για μεμονωμένες λέξεις που προσαρμόζονται στο περιεχόμενο των κειμένων. Αυτές οι ενσωματώσεις είναι στην πραγματικότητα οι εσωτερικές καταστάσεις ενός βαθέως νευρωνικού δικτύου το οποίο εκπαιδεύεται με αντικείμενα γλωσσικής μοντελοποίησης. Η εκπαίδευση των συμφραζομένων ενσωματώσεων πραγματοποιείται σε στάδιο προεκπαίδευσης, ανεξάρτητα από την κύρια εργασία. Το εκπαιδευμένο μοντέλο μπορεί στη συνέχεια να δημιουργήσει αναπαραστάσεις με βάση τα συμφραζόμενα για όλες τις λέξεις σε ένα δεδομένο κείμενο. Δύο τέτοια μοντέλα, είναι τα RNN και Transformer.

1.3 Μοντέλα NLP

Όπως προαναφέραμε, δύο μοντέλα που μπορούν να καταγράψουν τις εξαρτήσεις μεταξύ των λέξεων και χρησιμοποιούνται για την επεξεργασία φυσικής γλώσσας είναι τα RNN και Transformer. Τα RNN διαθέτουν την έννοια της

«μνήμης» που τους βοηθά να αποθηκεύουν τις καταστάσεις ή τις πληροφορίες προηγούμενων εισόδων για τη δημιουργία της επόμενης εξόδου της ακολουθίας. Ειδικά τα LSTM και GRUs διαθέτουν μηχανισμούς που στοχεύουν στην αντιμετώπιση του προβλήματος της μακροπρόθεσμης εξάρτησης.

Ωστόσο, τα RNN τείνουν να είναι αργά και η ικανότητά τους να μάθουν τις μακροπρόθεσμες εξαρτήσεις εξακολουθεί να είναι περιορισμένη. Το 2017 ένα εποπτευόμενο μοντέλο βαθιάς μάθησης, το οποίο ονομάστηκε Transformer, παρουσιάστηκε για πρώτη φορά και χρησιμοποιείται πλέον όλο και περισσότερο για προβλήματα NLP, αντικαθιστώντας τα μοντέλα RNN με μηχανισμούς προσοχής. Αυτό το χαρακτηριστικό επιτρέπει μεγαλύτερη παραλληλοποίηση από τα RNN και επομένως μειώνει τους χρόνους εκπαίδευσης. Επίσης προσδιορίζει το περιεχόμενο που προσδίδει νόημα σε κάθε λέξη της πρότασης. Στην συνέχεια, αναλύουμε την αρχιτεκτονική του Transformer καθώς και τα μοντέλα BERT, BioBERT και Longformer, που είναι βασισμένα στο Transformer.

1.3.1 Transformer

Το δίκτυο Transformer αποτελείται από έναν κωδικοποιητή καθώς και έναν αποκωδικοποιητή. Ο κωδικοποιητής κωδικοποιεί την ακολουθία εισόδου σε μία ενδιάμεση αναπαράσταση. Στη συνέχεια, η αναπαράσταση αυτή τροφοδοτείται στον αποκωδικοποιητή, ο οποίος παράγει την ακολουθία εξόδου. Ο μηχανισμός attention επιτρέπει στο μοντέλο σε κάθε βήμα να επικεντρώνεται σε εκείνο το τμήμα της ακολουθίας εισόδου, το οποίο θεωρεί σημαντικό για τον υπολογισμό της εξόδου. Με άλλα λόγια, το μοντέλο μαθαίνει να διακρίνει τις σχέσεις μεταξύ των συμβόλων της ακολουθίας εισόδου. Στην περίπτωση του Transformer η συνάρτηση attention είναι γνωστή ως scaled dot-product attention. Για κάθε σύμβολο x_i της ακολουθίας εισόδου υπολογίζονται τρία διανύσματα: το query q_i, το key k_i και το value v_i. Οι Q, K, V είναι οι πίνακες με όλα τα queries, keys, values αντίστοιχα ενώ d_k το μέγεθος των διανυσμάτων key και query.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{1.1}$$

Στο μοντέλο Transformer, τα διανύσματα query, key και value υπολογίζονται πολλές φορές με διαφορετικούς πίνακες προβολής, οι οποίοι μαθαίνονται κατά την εκπαίδευση του μοντέλου. Στη συνέχεια, ο μηχανισμός attention εφαρμόζεται παράλληλα για κάθε σύνολο διανυσμάτων και το τελικό αποτέλεσμα προκύπτει από την παράθεση των επιμέρους αποτελεσμάτων. Η τεχνική αυτή ονομάζεται multi-head attention και επιτρέπει στο μοντέλο να επικεντρώνεται ταυτόχρονα σε πολλά τμήματα της ακολουθίας εισόδου.

Ο κωδικοποιητής αποτελείται από έξι πανομοιότυπα επίπεδα τοποθετημένα σε στοίβα. Κάθε επίπεδο αποτελείται από δύο υποεπίπεδα: τον μηχανισμό multi-

head self-attention και ένα πλήρως συνδεδεμένο δίκτυο. Ο αποκωδικοποιητής αποτελείται και αυτός από έξι επίπεδα. Τα επίπεδα του αποκωδικοποιητή είναι όμοια με τα επίπεδα του κωδικοποιητή, με τη διαφορά ότι περιέχουν ένα επιπλέον υποεπίπεδο, το οποίο υλοποιεί τον μηχανισμό του multi-head attention μεταξύ της τρέχουσας κατάστασης του αποκωδικοποιητή και της εξόδου του τελευταίου κωδικοποιητή. Επίσης, ο μηχανισμός self-attention τροποποιείται, έτσι ώστε η έξοδος του αποκωδικοποιητή να εξαρτάται μόνο από τις προηγούμενες εξόδους. Αυτό επιτυγχάνεται με τη χρήση μασκών. Επίσης, επειδή το μοντέλο αυτό δεν χρησιμοποιεί ανάδραση, η κωδικοποίηση θέσης (positional encoding) είναι απαραίτητη για να κωδικοποιήσει την πληροφορία της θέσης του κάθε συμβόλου της ακολουθίας εισόδου.

1.3.2 BERT

Το BERT είναι ένα μοντέλο αναπαράστασης γλώσσας που κυκλοφόρησε το 2018 και η αρχιτεκτονική του είναι βασισμένη στα Transformer. Τα προεκπαιδευμένα μοντέλα BERT έχουν χρησιμοποιηθεί αποτελεσματικά σε διάφορες εργασίες επεξεργασίας φυσικής γλώσσας. Οι αναπαραστάσεις εισόδου και εξόδου του μοντέλου είναι συγκεκριμένες και έχουν μέγιστο μέγεθος 512 διακριτικά (tokens). Η είσοδος BERT μπορεί να είναι τόσο μια μεμονωμένη πρόταση όσο και ένα ζευγάρι προτάσεων. Το πρώτο διακριτικό κάθε ακολουθίας είναι πάντα ένα ειδικό διακριτικό ταξινόμησης ([CLS]) ενώ το τελευταίο διακριτικό κάθε πρότασης είναι ένα ειδικό διακριτικό ([SEP]). Το BERT, επίσης, προσθέτει στις ενσωματώσεις των tokens ενσωματώσεις τμημάτων ώστε να ενημερώσει το μοντέλο σε ποια πρόταση το toke ανήκει και ενσωματώσεις θέσης ώστε να ενημερώσει το μοντέλο πού στην ακολουθία ανήκει το token. Για την προεκπαίδευση του μοντέλου BERT χρησιμοποιήθηκαν η μοντελοποίηση μάσκας γλώσσας (Masked Language Model) και η πρόβλεψη επόμενης πρότασης (Next Sentence Prediction).

Μοντελοποίηση Μάσκας Γλώσσας

Προκειμένου να εκπαιδευτεί το BERT, το 15% των tokens εισόδου καλύπτονται τυχαία χρησιμοποιώντας ένα ειδικό token [MASK]. Το μοντέλο έτσι εκπαιδεύεται να προβλέπει αυτά τα tokens χρησιμοποιώντας όλα τα άλλα tokens της ακολουθίας. Ωστόσο, η εργασία fine-tuning δεν πρόκειται σε καμία περίπτωση να δει το token [MASK] στην είσοδό του. Έτσι, για να προσαρμόσει το μοντέλο, το 80% των περιπτώσεων, τα tokens του καλύπτονται κατα 15%, 10% των περιπτώσεων, τα tokens αντικαθίστανται κατά 15% με τυχαία tokens και το 10% των περιπτώσεων μένουν ως έχουν, δηλαδή ανέγγιχτα. Το πλεονέκτημα αυτής της διαδικασίας είναι ότι ο κωδικοποιητής δεν γνωρίζει ποιες λέξεις θα κληθεί να προβλέψει και ποιες έχουν αντικατασταθεί από τυχαίες

λέξεις, επομένως αναγκάζεται να διατηρήσει μια αναπαράσταση κατανομής με βάση τα συμφραζόμενα για κάθε token εισόδου.

Πρόβλεψη Επόμενης Πρότασης

Η πρόβλεψη της επόμενης πρότασης είναι σημαντική για την κατανόηση της σχέσης μεταξύ 2 προτάσεων. Το BERT, λοιπόν, είναι προεκπαιδευμένο για μια δυαδική εργασία πρόβλεψης επόμενης πρότασης. Πιο συγκεκριμένα, όταν επιλεχθούν 2 προτάσεις A και B για κάθε παράδειγμα προεκπαίδευσης, το 50% του χρόνου η B είναι η πραγματική επόμενη πρόταση που ακολουθεί το A (με την ετικέτα IsNext) και το άλλο 50% των φορών είναι μια τυχαία πρόταση από το σώμα (με την ένδειξη NotNext).

1.3.3 BioBERT

Το BioBERT είναι μια παραλλαγή του μοντέλου BERT και εισήχθη το 2019. Οι ερευνητές προεκπαίδευσαν το BERT σε βιοϊατρικά κείμενα, συγκεκριμένα σε περιλήψεις του PubMed και στα άρθρα πλήρους κειμένου PubMed Central. Το προεκπαιδευμένο μοντέλο αυτό είναι πολύ σημαντικό για την επεξεργασία της φυσικής γλώσσας στον χώρο της ιατρικής.

1.3.4 Longformer

Τα μοντέλα Transformer δεν μπορούν να επεξεργαστούν μεγάλες ακολουθίες λόγω της λειτουργίας της αυτοπροσοχής. Για την αντιμετώπιση αυτού του περιορισμού, το Longformer εισήχθη με ένα μηχανισμό προσοχής που κλιμακώνεται γραμμικά με το μήκος της ακολουθίας, καθιστώντας το εύκολο για την επεξεργασία εγγράφων χιλιάδων tokens ή περισσότερων.

1.4 Αναπαράσταση Γνώσης

Η αναπαράσταση γνώσης είναι το πεδίο της τεχνητής νοημοσύνης που σχετίζεται με την αναπαράσταση πληροφοριών για τον κόσμο με μια μορφή που ένας υπολογιστής μπορεί να χρησιμοποιήσει. Η αναπαράσταση γνώσης ενσωματώνει ευρήματα από τη λογική για να αυτοματοποιήσει διάφορα είδη συλλογισμών. Στις αναπαραστάσεις της γνώσης περιλαμβάνονται οι οντολογίες και τα γραφήματα γνώσης. Στην παρούσα διπλωματική ασχολούμαστε με την αναπαράσταση οντολογιών.

Μια οντολογία είναι ένα τυπικά καθορισμένο λεξιλόγιο για έναν συγκεκριμένο τομέα ενδιαφέροντος που χρησιμοποιείται για τη συλλογή γνώσεων σχετικά με αυτόν τον (περιορισμένο) τομέα ενδιαφέροντος. Η οντολογία, λοιπόν,

περιγράφει τις έννοιες στον τομέα αυτόν καθώς και τις σχέσεις μεταξύ αυτών των εννοιών. Η Γλώσσα Οντολογίας Ιστού (OWL) είναι μία από τις διασημότερες οικογένειες γλωσσών αναπαράστασης γνώσης για τη σύνταξη οντολογιών.

Η χρήση εξωτερικών πηγών γνώσης όπως οι οντολογίες και τα γραφήματα γνώσης μπορούν να χρησιμοποιηθούν για την παραγωγή ενσωματώσεων για τους όρους τους με στόχο τη σημασιολογική ενίσχυση ενός μοντέλου για την επεξεργασία φυσικής γλώσσας. Εστιάζουμε ιδιαίτερα στον αλγόριθμο OWL2Vec*, που χρησιμοποιήθηκε στην πειραματική μελέτη της διπλωματικής εργασίας για την παραγωγή των ενσωματώσεων της οντολογίας SNOMED CT.

Το OWL2Vec* αναπτύχθηκε το 2021 με σκοπό την δημιουργία ενός αλγορίθμου που θα μπορεί να κωδικοποιεί τη σημασιολογία μιας οντολογίας OWL λαμβάνοντας υπόψη τη δομή του γράφηματός της, τις λεξιλογικές πληροφορίες καθώς και τους λογικούς τελεστές. Το συνολικό πλαίσιο του OWL2Vec* αποτελείται κυρίως από δύο βασικά μέρη, την εξαγωγή πληροφοριών από την οντολογία με την παραγωγή τριών αρχείων (αρχείο δομής, λεξιλογικό αρχείο και συνδυαστικό) και την εκπαίδευση του γλωσσικού μοντέλου με τις πληροφορίες αυτές με σκοπό την παραγωγή ενσωματώσεων των οντοτήτων της οντολογίας.

Αρχικά, γίνεται η μετατροπή της αρχικής οντολογίας OWL σε ένα γράφημα σε RDF μορφή. Το OWL2Vec* ενσωματώνει δύο στρατηγικές για την διαδικασία αυτή. Η πρώτη στρατηγική εφαρμόζει τον μετασχηματισμό OWL to RDF Graph Mapping που ορίζεται από το W3C. Η δεύτερη στρατηγική βασίζεται σε κανόνες προβολής που προτείνονται στον Πίνακα 7.1. Στη συνέχεια δημιουργούνται τα τρία έγγραφα. Η δημιουργία του εγγράφου δομής στοχεύει στην αποτύπωση τόσο της δομής του γραφήματος όσο και των λογικών τελεστών της οντολογίας. Μια επιλογή είναι ο υπολογισμός τυχαίων περιπάτων για κάθε οντότητα-στόχο. Κάθε περίπατος, που είναι μια ακολουθία IRI οντοτήτων, αποτελεί πρόταση στο έγγραφο δομής. Το OWL2Vec* επιτρέπει επίσης τη χρήση του Weisfeiler Lehman kernel. Για να συλλάβει τους λογικούς τελεστές, το OWL2Vec* εξάγει όλα τα αξιώματα της οντολογίας και συμπληρώνει τις προτάσεις του εγγράφου δομής. Το λεξιλογικό έγγραφο περιλαμβάνει προτάσεις που δημιουργούνται από τις προτάσεις IRI οντοτήτων στο έγγραφο δομής και προτάσεις που εξάγονται από τα σχετικά αξιώματα λεξιλογικού σχολιασμού στην οντολογία. Το OWL2Vec*, επίσης, εξάγει ένα συνδυαστικό έγγραφο από το έγγραφο δομής και τους λεξιλογικούς σχολιασμούς της οντότητας, έτσι ώστε να διατηρηθεί η συσχέτιση μεταξύ των οντοτήτων και των λέξεων στις λεξιλογικές πληροφορίες. Αυτό μπορεί να προσθέσει θόρυβο στη συσχέτιση μεταξύ των λέξεων και να επηρεάσει αρνητικά τις ενσωματώσεις των λέξεων. Στο τέλος, το OWL2Vec* συγχωνεύει τα τρία έγγραφα σε ένα έγγραφο και, στη συνέχεια, χρησιμοποιεί αυτό το έγγραφο για να εκπαιδεύσει ένα μοντέλο Word2vec. Μπορούμε επίσης να εκπαιδεύσουμε εκ των προτέρων το μοντέλο Word2vec αλλά αυτό μπορεί να αποδειχθεί θορυβώδες. Με το εκπαιδευμένο μο-

ντέλο ενσωμάτωσης λέξεων, το OWL2Vec* υπολογίζει τις ενσωματώσεις κάθε IRI και κάθε λέξης.

1.5 Πειραματική Μελέτη

Στην πειραματική μελέτη της διπλωματικής αυτής συγκρίνουμε την απόδοση των προεκπαιδευμένων μοντέλων BERT με τον σημασιολογικό εμπλουτισμό τους με την χρήση της οντολογίας SNOMED CT. Εστιάζουμε την μελέτη μας στα μοντέλα BERT και BioBERT. Για την αξιολόγση χρησιμοποιούμε ένα υποσύνολο από τα έγγραφα της OHSUMED και συγκεκριμένα τον τίτλο και την περίληψή τους. Επίσης, χρησιμοποιούμε το εργαλείο MetaMap ώστε να αντιστοιχίσουμε τα ιατρικά κείμενα με όρους της SNOMED CT.

Αρχικά, επικεντρωνόμαστε στο πρόβλημα της ανάκτησης πληροφορίας από ιατρικά έγγραφα. Χρησιμοποιούμε το OWL2Vec* ώστε να εμπλουτίσουμε τις ενσωματώσεις των μοντέλων BERT με ενσωματώσεις της οντολογίας SNOMED CT. Παράγουμε τις ενσωματώσεις κάθε κειμένου και κάθε ερώτησης προσθέτοντας τις ενσωματώσεις τους από το BERT και από τον μέσο όρο των ενσωματώσεων των όρων τους από το OWL2Vec*. Στη συνέχεια, ταξινομούμε τη συνάφεια των εγγράφων για καθένα ερώτημα υπολογίζοντας την απόσταση μεταξύ δύο διανυσματικών αναπαραστάσεων για την ανάκτηση εγγράφων που σχετίζονται με το ερώτημα. Η απόσταση μεταξύ του διανύσματος κειμένου d_i και του ερωτήματος q_j υπολογίζεται με το cosine similarity.

$$\text{relevance}(d_i, q_j) = \frac{d_i \cdot q_j}{\|d_i\| \cdot \|q_j\|} \tag{1.2}$$

Πειραματιζόμαστε με διαφορετικές ρυθμίσεις του OWL2Vec* καθώς και τροποποιήσεις του συνόλου δεδομένων μας, χωρίζοντας την περίληψη των κειμένων σε μέρη, και της οντολογίας SNOMED CT, εισάγοντας και τα κείμενα μέσα στην οντολογία ώστε να παραχθούν οι ενσωματώσεις κάθε κειμένου αυτόματα. Για την αξιολόγηση του συστήματός μας χρησιμοποιούμε την μετρική $nDCG$, που ορίζεται για k in {0,1,...,N}:

$$\text{nDCG}_k = \frac{1}{Q}\sum_{q=1}^{Q}\frac{IDCG_k^{(q)}}{DCG_k^{(q)}}, \qquad \text{for} \quad \text{DCG}_k^{(q)} = rel_1^{(q)} + \sum_{i=2}^{k}\frac{rel_i^{(q)}}{\log_2(i)} \tag{1.3}$$

όπου $IDCG$ υποδηλώνει το ιδανικό και υψηλότερο DCG και το $rel_i^{(q)}$ αναφέρεται στην συνάφεια του i^{th} αποτελέσματος που κατατάσσεται ανάλογα με το ερώτημα q.

Μετά την εκτέλεση όλων των πειραμάτων βλέπουμε ότι το OWL2Vec* αδυνατεί να αποτυπώσει σωστά την οντολογία SNOMED CT καθώς είναι μια μεγάλη οντολογία με πολλούς όρους. Τα καλύτερα αποτελέσματα τα λαμβάνουμε χωρίζοντας την περίληψη των κειμένων σε μέρη και εισάγοντας τα στην οντολογία SNOMED CT. Η βελτίωση όμως δεν είναι τόσο μεγάλη συγκριτικά με τα απλά μοντέλα BERT.

Την αδυναμία αυτή του OWL2Vec* την αποδεικνύουμε και στο πρόβλημα της ταξινόμησης. Και εκεί η επαύξηση των ενσωματώσεων των μοντέλων BERT με τις ενσωματώσεις της οντολογίας SNOMED CT επιδρά αρνητικά στην ταξινόμηση των κειμένων. Γι΄ αυτό, αποφασίζουμε να δημιουργήσουμε ένα άλλο σύστημα που θα αξιοποιούμε την ιεραρχία των εννοιών της SNOMED CT. Συγκεκριμένα, δημιουργούμε μια μέθοδο φιλτραρίσματος των κειμένων. Επικεντρωνόμαστε σε δύο κλάσεις: Μυοσκελετικές Ασθένειες και Ενδοκρινικές Ασθένειες. Αναζητούμε τους όρους που σχετίζονται με τις έννοιες αυτές στην SNOMED CT καθώς και τα συνώνυμά τους. Επίσης, αναζητούμε τους γονείς και τα παιδιά τους, τους γονείς των γονέων και τα παιδιά των παιδιών κοκ. Έτσι, διερευνούμε διαφορετικά βάθη της οντολογίας. Με τις έννοιες που βρίσκουμε, φιλτράρουμε τα κείμενά μας χρησιμοποιώντας διαφορετικά κατώφλια και αξιολογούμε το σύστημά μας χρησιμοποιώντας την μετρική f1-score.

Διαπιστώνουμε ότι ένα απλό φιλτράρισμα καταφέρνει να προσεγγίσει αρκετά την απόδοση του μοντέλου BERT. Στη συνέχεια συνδυάζουμε το φιλτράρισμα που προαναφέραμε με ένα νευρωνικό δίκτυο με 1 κρυφό επίπεδο. Σαν είσοδο χρησιμοποιούμε τις ενσωματώσεις του μοντέλου BERT. Διαπιστώνουμε ότι το μοντέλο BERT συνδυασμένο με το φιλτράρισμα ξεπερνά την απόδοση του απλού BioBERT, το οποίο είναι ένα εξειδικευμένο μοντέλο για ιατρικά κείμενα. Έτσι, αποδεικνύουμε ότι μια οντολογία μπορεί να εξειδικεύσει ένα γενικευμένο μοντέλο ώστε να φέρει καλύτερα αποτελέσματα.

Chapter 2

Introduction

This chapter's goal is to describe the motivation behind this work and introduce readers to the actual problem. It also contains previous relevant work, as well as a brief outline of the rest of the thesis and its contribution.

2.1 Motivation

Information retrieval was always a challenging task in natural language processing, especially when that concerns the medical sector. Imagine a researcher who needs to find information concerning a specific disease among multiple documents. We understand that navigating existing and upcoming literature with efficient and fast systems can not only make researcher's life easier but also lead to the improvement of medical research. Indeed, this need is expressed on information retrieval and question answering task as well as classification task.

Lately, the need to apply machine learning models in the biomedical natural language processing field has been more and more increased. Indeed, deep neural architectures such as BERT-based models have shown great potential in information retrieval and classification, rendering them strong vanilla models. However, while machine learning methods keep improving, the domain topics change rapidly and so do the related textual resources (scientific publications, reports, clinical trials). Thus, although standard datasets provide a solid basis for training, improving and evaluating new methods, they cannot account for emerging topics, new entities and terminology.

Knowledge bases, such as SNOMED CT, consists of a huge number of medical terms and in fact the specific terminology is updated every six months and so it includes each time more and more new entities. As a result, we wanted to explore the potential of using such knowledge sources in a

post-processing manner in order to enhance such pretrained models.

2.2 Related Work

While there is a range of work that relates to the work presented in this thesis, the main line of research is the semantinc enrichment of natural language processing models with the use of external knowledge sources. The specific diploma thesis was mainly inspired by Dervakos et al. [15]. Dervakos et al. proposed a filtering method in information retrieval task based on concept hierarchy of SNOMED CT. In particular, they enhanced queries and documents with SNOMED CT concepts, imposed filters on concept co-occurence between them and in that way they enriched bert-based models. This approach showed competitive performance when applied on medical papers. Similarly, we wanted to investigate the semantic enrichment of these models with owl embeddings and also extend this approach in another task.

Especially in textual classification significant performance boosts can be obtained by using external knowledge sources to complement the textual representations and provide more informative features. Ostendorf et al. [44] proposed the enrichment of BERT models with knowledge graphs embeddings which encode author information for the classification of books. Zhang et al. [69] experimented with external knowledge graphs to enrich embedding information in order to ultimately improve language understanding. They used structural knowledge represented by Wikidata entities and their relation to each other. Earlier, Wang et al. [61] proposed and evaluated an approach to improve text classification with knowledge from Wikipedia. Based on a bag of words approach, they derived a thesaurus of concepts from Wikipedia and used it for document expansion. The resulting document representation improved the performance of an SVM classifier for predicting text categories. Ritchie et al [48] used owl embeddings from OWL2Vec* to drive the computation of ontology entity clusters.

Focusing on the information retrieval task, several publications used external knowledge sources to improve their performance. Agosti et al. [3] considered the relation between text and queries and aimed to reduce the semantic gap between queries and documents, by incorporating polysemy and synonymy information during the training of neural networks.

2.3 Thesis Contribution

In this thesis, we focus on different ways to enhance bert-based embeddings with knowledge extracted from SNOMED CT. BERT (Bidirectional Encoder Representations from Transformers) is a family of high performance pre-trained language models which produce state-of-the-art results in a wide variety of NLP tasks [16]. BERT's key technical innovation is applying the bidirectional training of Transformers [60] to language modelling. By using multiple attention mechanisms (multi-head attention), the model is able to capture a broader range of relationships between words than would be possible with a single attention mechanism. Moreover, BERT stacks multiple layers of attention, each of which operates on the output of the layer that came before. Through this repeated composition of word embeddings, BERT is able to form very rich representations as it gets to the deepest layers of the model. As for the knowledge source we choose SNOMED CT, which is the most comprehensive clinical healthcare terminology, consisting of more than 350,000 concepts and covering clinical findings, symptoms, diagnoses, procedures, body structures, organisms and other etiologies, substances, pharmaceuticals, devices and specimens among others.

We concentrate on two different tasks of natural language processing: information retrieval and text classification. For evaluation we use subsets of OHSUMED dataset to assess the improvement our methods can achieve compared to different BERT models, used as baselines. Firstly, for both tasks, we enrich bert-based embeddings with owl embeddings from SNOMED CT. The owl embeddings for each document are produced using the framework OWL2Vec*, which preserves the lexical information and the logical constructors of an ontology. More specifically, we search the terms of each document in SNOMED CT and generate their embeddings with OWL2Vec*. For information retrieval, given a query we retrieve its bert-based and owl embeddings and we calculate the cosine similary of this vector representation between query and document. We experiment with different settings of OWL2Vec* and different pre-processes of our dataset and ontology and we prove that OWL2Vec* is currently unable to be applied in large vocabularies, like SNOMED CT. We use, also, this semantic enrichment of embeddings on text classification while using a neural network with 1 hidden layer to classify our documents. Our assumptions for the inefficiency of OWL2Vec* for large ontologies are again confirmed.

We then apply a filtering method on documents for the classification task. We create a concept filter on documents which is based on concept co-occurence between the concepts associated with the class and the terms of the documents. For finding the concepts that are associated with the class

we take advantage of the concept hierarchy of SNOMED CT. We experiment with different depths on SNOMED CT and different thresholds on documents. We prove that by removing the documents with no concept co-occurrence with the class and by creating a neural network, as before, with 1 hidden layer and the bert-based embeddings as input, we achieve better performance than unfiltered bert-based models. As a result, we prove that a specialized ontology, can successfully be applied to adapt out-of-domain models to a new domain and improve the performance of natural language processing models.

2.4 Thesis Structure

The remaining of the thesis is structured as follows. In Chapter 3 we make an introduction to artificial intelligence and natural language processing (NLP) tasks. In Chapter 4 we present the background of deep neural networks, which is helpful for completely understanding how the models that we use in our experimental study work. We, also, present some fundamental neural networks that are commonly used in NLP and the basic ideas behind them. Within Chapter 5 we describe several methods for representing language in computers and we emphasize on word embedding models. We continue in Chapter 6 by introducing the Transformer Model and we focus on bert-based models, on which our experiments are based. In the next chapter (Chapter 7), we emphasize on knowledge representation and how external knowledge sources such as ontologies and knowledge graphs can be used for producing vector representations that can semantically enhance a NLP model. Finally, in Chapter 8 we present the experimental results of our framework and in Chapter 9 we make our concluding remarks.

Chapter 3

Introduction to Artificial Intelligence

In our days, especially during the last decade, the main focus is on Artificial Intelligence, whose practical applications and current research topics are increasing and expanding. It was founded as an academic discipline in 1956, and in the years since has experienced several waves of optimism, followed by disappointment and the loss of funding (known as an "AI winter"). In the first steps of Artificial Intelligence, its goal was to solve problems which were challenging and intellectually hard for human beings, like complex mathematical operations, with impressive speed and accuracy. What's proven really hard though, is to solve problems that are easy, like automatic, for human brain, and that they can't be formally described, such as recognizing similar faces in images or spoken words [25].

In order for this kind of problems to be solved, machines need to learn from experience and examples. The procedure of running algorithms, that improve the ability of a computer to solve a specific problem through experience and by the use of data is called Machine Learning.

For more than 50 years computer scientists focus on developing the ability of computer programs to understand human language as it is spoken and written, referred to as natural language. This subfield of Linguistics, Computer Science and Artificial Intelligence is called Natural Language Processing (NLP).

In the following sections we give a definition of Machine Learning algorithms (Section 3.1), which can be approached by four different methods: supervised learning (Section 3.1.1, unsupervised learning (Section 3.1.2), semi-supervised learning (Section 3.1.3) and reinforcement learning (Section 3.1.4). Furthermore we analyze Natural Language Processing and its applications (Section 3.2).

3.1 Machine Learning

Machine learning is the study of computer algorithms that can improve automatically through experience and by the use of data. It is seen as a subset of Artificial Intelligence. Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions in new cases. They are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

The discipline of Machine Learning employs various approaches to teach computers to accomplish tasks where no fully satisfactory algorithm is available. Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the "signal" or "feedback" available to the learning system.

3.1.1 Supervised Learning

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data, and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.

Types of supervised learning algorithms include active learning, classification and regression. Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range. Similarity learning is an area of supervised machine learning closely related to regression and classification, but the goal is to learn from examples using a similarity function that measures how similar or related two objects are. It has applications in ranking, recommendation systems, visual identity tracking, face verification, and speaker verification [63].

3.1.2 Unsupervised Learning

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention.

Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, image and pattern recognition. It's also used to reduce the number of features in a model through the process of dimensionality reduction; principal component analysis (PCA) and singular value decomposition (SVD) are two common approaches for this [19].

3.1.3 Semi-supervised Learning

Semi-supervised learning is an approach to machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning (with no labeled training data) and supervised learning (with only labeled training data).

Unlabeled data, when used in conjunction with a small amount of labeled data, can produce considerable improvement in learning accuracy. The acquisition of labeled data for a learning problem often requires a skilled human agent or a physical experiment. The cost associated with the labeling process thus may render large, fully labeled training sets infeasible, whereas acquisition of unlabeled data is relatively inexpensive [62].

3.1.4 Reinforcement Learning

Reinforcement learning is a machine learning technique, whose goal is to solve the problem of choosing the most suitable action to take in a given environment, in order to maximize a specific reward. In contrast to traditional supervised learning, reinforcement learning does not get desired labels of the training inputs, but estimates the best outputs (actions), based on the reward it yields. Often, the current action not only affects the immediate reward, but the long-term reward as well. An important issue of reinforcement learning is the trade-off between exploration, in which the system experiments with new kinds of actions to check how profitable or harmful they are, and exploitation, in which the system follows a more conservative approach of selecting already known actions to achieve a high reward. The most efficient approach is to

maintain a balance between the two strategies, because focusing on only one of those will result in poor outcomes [7].

Reinforcement learning is mostly used to train computers to play games and they can achieve really high scores compared to human performance. Reinforcement learning remains an active area of machine learning research.

3.2 Natural Language Processing

Natural Language Processing has its roots in the 1950s. Already in 1950, Alan Turing published an article titled "Computing Machinery and Intelligence" which proposed what is now called the Turing test as a criterion of intelligence, though at the time that was not articulated as a problem separate from artificial intelligence. The proposed test includes a task that involves the automated interpretation and generation of natural language.

Nowadays, representation learning and deep neural network-style machine learning methods became widespread in natural language processing. That popularity was due partly to a flurry of results showing that such techniques can achieve state-of-the-art results in many natural language tasks, e.g., in language modeling and parsing. This is increasingly important in medicine and healthcare, where NLP helps analyze notes and text in electronic health records that would otherwise be inaccessible for study when seeking to improve care.

In NLP, there are two main phases: data preprocessing and algorithm development. Data preprocessing involves preparing and "cleaning" text data for machines to be able to analyze it. It specifically includes methods such as tokenization (the split of the text into smaller units) and stop word removal (the removal of the most common words from the text). Once the data has been preprocessed, an algorithm is developed to process it. There are many different natural language processing algorithms, but two main types are commonly used: rules-based systems which uses carefully designed linguistic rules and machine learning-based system [64].

The following is a list of some of the most commonly researched tasks in natural language processing.

- **Information Retrieval:** With the help of NLP, we can find the needed piece among unstructured data. An information retrieval system indexes a collection of documents, analyzes the user's query, then compares each document's description with the query and presents the relevant results.

- **Information grouping:** Grouping, or text classification, is performed via the text tags. The NLP model is trained to classify documents according to specific attributes: subject, document type, time, author, language, etc. Text classification usually requires labeled data. Information grouping is used for supervised machine learning, which correspondingly triggers a multitude of use cases.

- **Machine Translation:** This typically involves translating one natural language into another, preserving the meaning and producing fluent text as a result. Different methods and approaches are used here: rule-based, statistical and neural machine translation.

- **Summarization:** NLP algorithms can be used to create a shortened version of an article, document, number of entries, etc., with main points and key ideas included. There are two general approaches: abstractive and extractive summarization. In the first case, the NLP model creates an entirely new summary in terms of phrases and sentences used in the analyzed text. In the second case, the model extracts phrases and sentences from the existing text and groups them into a summary.

- **Sentiment analysis:** It's a type of text classification where the NLP algorithms determine the text's positive, negative, or neutral connotation. Use cases include analyzing customers' feedback, detecting trends, conducting market research, etc., via an analysis of tweets, posts, reviews and other reactions. Sentiment analysis can encompass everything from the release of a new game on the App Store to political speeches and regulation changes.

- **Named-Entity Recognition:** NER is an entity extraction, identification and categorization. It involves extracting names of locations, people and things from the text and placing them under certain categories – Person, Company, Time, Location, etc. The use cases may include content classification for SEO, customer support, patient lab reports analysis, academic research and others.

- **Automated speech recognition (ASR):** NLP techniques are actually designed for text but can also be applied to spoken input. ASR transcribes oral data into a stream of words. Neural networks and hidden Markov models are used to reduce speech recognition's error rate, however, it's still far from perfect. The main challenge is the lack of segmentation in oral documents. And while human listeners can easily segment spoken input, the automatic speech recognizer provides unannotated output.

As a result, we understand that the value of using NLP techniques is apparent, and the application areas for natural language processing are numerous. But so are the challenges researchers are facing to make NLP results resemble human output.

Chapter 4

Deep Learning Background

During the last decades, the field of Machine Learning has brought forth a variety of remarkable advancements in sophisticated learning algorithms and efficient pre-processing techniques. One of these advancements was the evolution of artificial neural networks (ANNs) towards increasingly deep neural network architectures with improved learning capabilities summarized as Deep Learning.

The idea is to employ Machine Learning to not only train the computer to map the input feature vector to a desired output, but also to learn the representation itself. Towards this end, Deep Learning can give the solution to extracting high-level features from raw data by introducing representations that are expressed in terms of other, simpler representations. So Deep Learning enables the computer to build complex concepts out of simple ones. Deep Learning is actually a subfield of Machine Learning inspired by the structure and function of the human brain and the way humans think [32] [25].

During the 1960s, the interest in neural networks started with the development of learning machines called perceptrons, that imitated the way brain neurons work. However, as perceptrons could not guarantee successful results in case of non-linearly separable data, the idea was then to employ multilayers of perceptrons. This multilayer training is referred to as Deep Learning, and its practical implementations are mostly associated with large data sets.

Deep Learning does not always provide the best possible solution; there are numerous applications that are better handled by more traditional methods. However, it has been proven extremely useful in applications that have been challenging for other methods. In fact, it has offered the opportunity to solve many problems in various domains and has been used in fields like natural language processing and understanding [24].

In the following sections we begin with analyzing the function of a single perceptron (Section 4.1), we then describe how they are combined to

create multilayer neural networks and more specifically we focus on their learning process (Section 4.2). Finally, we present Recurrent Neural Networks (Section 4.3), which are commonly used in natural language processing.

4.1 The Perceptron

In 1958 the American psychologist Frank Rosenblatt, inspired by the Hebbian theory of synaptic plasticity (i.e. the adaptation of brain neurons during the learning process), came up with the perceptron.

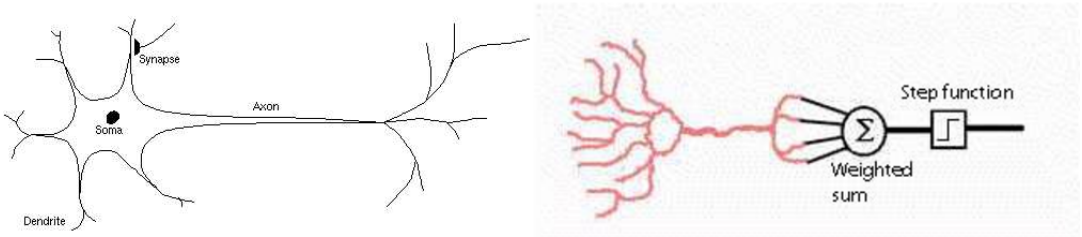


Figure 4.1: (left) A Biological Neuron. (right) A Perceptron. Source: Neural Networks: The perceptron (Fig. 1 in [10]).

The perceptron is a mathematical model of a biological neuron. While in actual neurons the dendrite receives electrical signals from the axons of other neurons, in the perceptron these electrical signals are represented as numerical values.

At the synapses between the dendrite and axons, electrical signals are modulated in various amounts. This is also modeled in the perceptron by multiplying each input value x_i by a value called the weight w_i.

An actual neuron fires an output signal only when the total strength of the input signals exceed a certain threshold. This threshold is called bias b. The total strength of the input signals is modeled as the weighted sum of the input values [10]. The weighted sum is expressed as follows:

$$w_1x_1 + w_2x_2 + ... + w_nx_n + b = \sum_{i=1}^{n} w_ix_i + b = \boldsymbol{w}^\top \boldsymbol{x} + b \tag{4.1}$$

At its simplest a step function is applied on the sum (4.1) to determine whether the neuron will fire or not. This function was originally used by Rosenblatt [10]. The output f of the neuron can be described by the following equation:

$$f(x) = \begin{cases} 1, & \text{if } \boldsymbol{w}^\top \boldsymbol{x} + b >= 0 \\ 0, & \text{if } \boldsymbol{w}^\top \boldsymbol{x} + b < 0 \end{cases} \tag{4.2}$$

4.1.1 Perceptron Training Algorithm

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

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0 \tag{4.3}$$

This means that we can employ a single perceptron unit to solve a classification problem by learning this linear boundary between linearly separable pattern classes.

An example of the Perceptron Training Algorithm, which according to the Perceptron Convergence Theorem will surely converge to a solution (a set of weights that define a hyperplane) after a finite number of steps, if the pattern classes are linearly separable, is explained below. We first define some variables:

• the learning rate r of the perceptron which is between 0 and 1 while larger values make the weight changes more volatile.

• $f(\boldsymbol{z})$ denotes the output from the perceptron for an input vector $\boldsymbol{z}$.

• $D = (\boldsymbol{x}_1, d_1), ..., (\boldsymbol{x}_s, d_s)$ is the training set of s samples where x_j is the n-dimensional input vector and d_j the desired output value of the perceptron for that input.

• $x_{j,i}$ is the value of the i_{th} feature of the j_{th} training input vector.

• $x_{j,0} = 1$.

• w_i is the i_{th} value in the weight vector, to be multiplied by the value of the i_{th} input feature.

So the steps in the algorithm are:

1 Initialize the weights. Weights may be initialized to 0 or to a small random value.

2 For each example j in our training set D, perform the following steps over the input x_j and desired output d_j:

   a Calculate the actual output: $y_j(t) = f[\boldsymbol{w}(t) \cdot \boldsymbol{x}_j] = f[w_0(t)x_{j,0} + w_1(t)x_{j,1} + ... + w_n(t)x_{j,n}]$.

   b Update the weights: $w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t))w_{j,i}$ for all features $0 \leq i \leq n$.

The concept behind this algorithm is that if a pattern is misclassified, we are trying to shift the weight vector to a direction that increases the probability of correct classification the next time the specific pattern is presented. That's also why if the classification of a pattern is correct, no change is applied to the weight vector [65].

4.2 Multilayer Feed-Forward Neural Networks

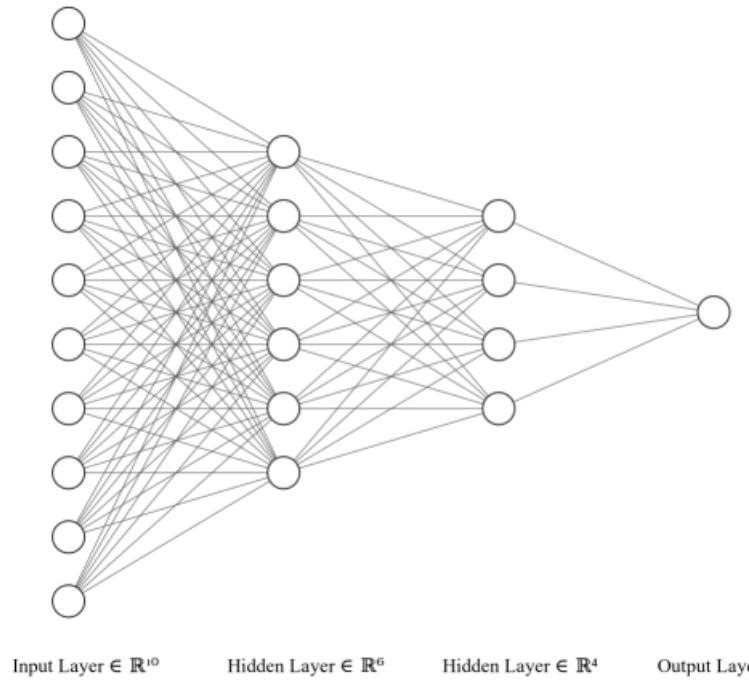


Figure 4.2: Architecture of an MLP which is constituded by an input layer with 10 neurons, two hidden layers with six and four neurons respectively and an output layer with one neuron. Source: (Fig. 1 in [23]).

In 1996 Minsky and Papert showed that a single layer Feed-Forward Neural Network (FFN) cannot solve problems in which the data is not linearly separable, such as the XOR problem [42]. The solution is to add one or more hidden layers to FNN. Per Universal Approximation Theorem, a FNN with one hidden layer can represent any function, although in practice training such a model is very difficult (if not impossible), hence, we usually add multiple hidden layers to solve complex problems [13]. These are called the Multilayer Feed-Forward Neural Networks.

In a multi-layer neural network, we have an input layer, an output layer, and one or more hidden layers (between input and output layers). The input layer has as many neurons as the dimension of the input data. The number of neurons in the output layer depends on the type of the problem the neural network is trying to solve. The more hidden layers that we have (and the more neurons we have in each hidden layer), our neural network can estimate more complex functions [5] [35].

4.2.1 Activation Functions

An activation function is applied on the output and hidden layer to determine whether the neuron will fire or not. However, the activation function described in (4.2) is a hard thresholding function which is discontinuous at 0 and causes problems in mathematical computations. Thus, it cannot be used in multi-layers networks. We can overcome this problem by using smoother versions of the above function.

In all cases, let z denote the output of the perceptron before thresholding and h the function employed to calculate the neuron's final output.

The sigmoid function is used very frequently as the activation function of artificial neurons. It can be used both at the output layer and hidden layers of a multilayer network and it allows the network to model non-linear relationships between input and output.

$$h(z) = \frac{1}{1 + e^{-z}} \tag{4.4}$$

Hyperbolic tangent, similarly to Sigmoid function, is a soft step function. But its range is between -1 and 1 (instead of 0 and 1).

$$h(z) = \tanh z = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{4.5}$$

ReLU (Rectified Linear Unit) is an activation function popular in deep neural networks. ReLU keeps all the positive inputs unchanged and sets all negative inputs to zero:

$$h(z) = \max(0, z) \tag{4.6}$$

The softmax function can convert a K dimension vector to another k dimension vector. After applying softmax, each component will be in the interval (0,1) and the sum of components will be 1.

$$h(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4.7}$$

4.2.2 Learning Process

Machine learning algorithms work on the basis of trying to learn from experience and training examples by optimizing a cost function. In order to train a neural network, there are four main steps that should be followed:

1 Forward Propagation: given an input, the model calculates an output.

2 Backward Propagation: this step calculates the gradient of the output and how each weight in the model affects it. This will then be used in the optimizer step.

3 Loss Function: a cost function is defined in order to know the accuracy of the model. It's a metric that shows as how far is the model from the correct label.

4 Optimization: tries to minimize the cost function using the gradient calculated in the backward propagation step by adjusting the weights of the model.

4.2.2.1 Backpropagation Algorithm

Neural network training relies on repeated application of the Backpropagation Algorithm to compute gradients of the loss with respect to model parameters. These gradients are then used to update the parameters of the model using the gradient-based optimization strategy. The Backpropagation Algorithm consists of two phases: the forward and backward pass [59].

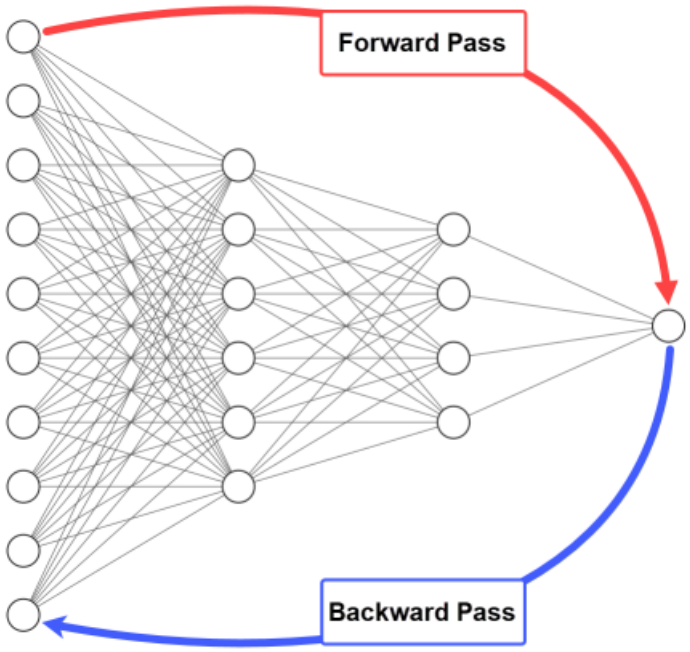


Figure 4.3: Architecture of an MLP where the forward and the backward pass is illustrated. Source: (Fig. 2 in [23]).

In the forward pass, the input data gets propagated to the input layer, goes through the hidden layer(s), then the network's predictions from the output layer are measured and finally the network error is calculated based on the predictions the network made. The process of propagating the inputs from the input layer to the output layer is called forward propagation. Once the network error is calculated, then the forward propagation phase has ended, and backward pass starts [23].

In the backward pass, the flow is reversed so that the error is propagated from the output layer to the input layer after passing through the hidden layer(s). The process of propagating the network error from the output layer to the input layer is called backward propagation, or simple backpropagation. The backpropagation algorithm is the set of steps used to update network weights to reduce the network error [23].

Let consider a multilayer perceptron (MLP) with an input layer h_0, followed by L hidden layers $h_1, h_2, ..., h_L$, followed by an output layer O. Except of the input layer, each layer is described by a set of parameters, the weights and biases, as $h_l = \{\boldsymbol{W}_l, b_l\}$, for all $l = 1, ..., L$. The final output layer is also governed by the weight and bias parameters $O = \{\boldsymbol{W}_o, b_o\}$. We use the element-wise activation function ϕ in our model for each hidden layer [59].

The forward propagation phase:
In forward propagation pass there are 4 main steps:

1 Compute the output of the first hidden layer as a function of the input $\boldsymbol{x}$:

$$\boldsymbol{h}_1 = \phi(\boldsymbol{W}_1^\top \boldsymbol{x} + b_1) \tag{4.8}$$

2 For each hidden layer, compute the output of that layer as a function of the output of the previous layer as:

$$\boldsymbol{h}_l = \phi(\boldsymbol{W}_l^\top \boldsymbol{h}_{l-1} + b_l), \quad \forall l = 2, ..., L \tag{4.9}$$

3 Finally, compute the output of the neural network as:

$$o = \boldsymbol{W}_o^\top \boldsymbol{h}_L + b_o \tag{4.10}$$

4 Compute the loss of the model as the error of the predicted output o compared to the desired output y as $l(o, y)$.

The backward propagation phase:
After the forward pass is complete, the backward pass starts:

1 Compute gradient of loss with respect to the output and store it for use in steps 2 and 3. :

$$\nabla_o^{l(o,y)} = \frac{\partial l(o,y)}{\partial o} \tag{4.11}$$

2 Compute gradients of the loss with respect to the parameters of the output layer.

$$\nabla_{\boldsymbol{W}_o}^{l(o,y)} = (\frac{\partial o}{\partial \boldsymbol{W}_o})^\top \nabla_o^{l(o,y)} \tag{4.12}$$

$$\nabla_{b_o}^{l(o,y)} = (\frac{\partial o}{\partial b_o})^\top \nabla_o^{l(o,y)} \tag{4.13}$$

3 Compute gradient of the output of the outermost hidden layer H_L with respect to the loss and store it for use in step 4 and 5.

$$\nabla_{\boldsymbol{h}_L}^{l(o,y)} = (\frac{\partial o}{\partial \boldsymbol{h}_L})^\top \nabla_o^{l(o,y)} \tag{4.14}$$

4 Compute gradients of the loss with respect to the parameters of the outermost hidden layer H_L.

$$\nabla_{\boldsymbol{W}_L}^{l(o,y)} = (\frac{\partial \boldsymbol{h}_L}{\partial \boldsymbol{W}_L})^\top \nabla_{\boldsymbol{h}_L}^{l(o,y)} \tag{4.15}$$

$$\nabla_{b_L}^{l(o,y)} = (\frac{\partial \boldsymbol{h}_L}{\partial b_L})^\top \nabla_{\boldsymbol{h}_L}^{l(o,y)} \tag{4.16}$$

5 Let $l = L - 1$. Compute and store gradients of outputs and parameters of that layer using gradients of layer $l + 1$ as follows.

$$\nabla_{\boldsymbol{h}_l}^{l(o,y)} = (\frac{\partial \boldsymbol{h}_{l+1}}{\partial \boldsymbol{h}_l})^\top \nabla_{\boldsymbol{h}_{l+1}}^{l(o,y)}, \quad \forall l = 1, ..., L-1 \tag{4.17}$$

$$\nabla_{\boldsymbol{W}_l}^{l(o,y)} = (\frac{\partial \boldsymbol{h}_l}{\partial \boldsymbol{W}_l})^\top \nabla_{\boldsymbol{h}_l}^{l(o,y)}, \quad \forall l = 1, ..., L-1 \tag{4.18}$$

$$\nabla_{b_l}^{l(o,y)} = (\frac{\partial \boldsymbol{h}_l}{\partial b_l})^\top \nabla_{\boldsymbol{h}_l}^{l(o,y)}, \quad \forall l = 1, ..., L-1 \tag{4.19}$$

6 If $l = 1$, terminate. Else, set $l = l - 1$. Go back to step 5.

4.2.2.2 Loss Function

The cost function is also called the loss function and is task-dependent. It has the aim of objectively measuring how much the network's predicted output is different than the expected output. We will now go through some basic loss functions, that are widely used in machine learning. Loss functions can be classified into two major categories depending upon the type of learning task we are dealing with, Regression losses and Classification losses.

Regression Loss Functions

Regression predictive modeling is the task of approximating a mapping function from input variables to a continuous output variable.

- Squared Error Loss: Squared Error Loss is the squared difference between the actual value y and the predicted o.

$$l(o, y) = (y - o)^2 \tag{4.20}$$

- Absolute Error Loss: Absolute Error Loss is the absolute value of the difference between the actual y and the predicted value o.

$$l(o, y) = |y - o| \tag{4.21}$$

Classification Loss Functions

Classification predictive modeling is the task of approximating a mapping function from input variables to discrete output variables.

- Cross Entropy Loss: Cross Entropy Loss increases as the predicted probability diverges from the actual label.

$$l(o, y) = -(y \log(o) + (1 - y) \log(1 - o)) \tag{4.22}$$

- Hinge Loss: Hinge Loss not only penalizes the wrong predictions but also the right predictions that are not confident.

$$l(o, y) = \max(0, 1 - y * o) \tag{4.23}$$

4.2.2.3 Optimization

Optimization is the process of choosing the most suitable model parameters in order to minimize the loss function, which is scalar. This problem of minimizing a loss function $l(\boldsymbol{a})$ with respect to a parameter vector a can be solved by a gradient descent procedure. The idea behind this iterative

method is to use gradient information from the loss function, in order to update the parameter vector by comprising a small step in the direction of negative gradient. In general $\boldsymbol{a}(k+1)$ is calculated from $\boldsymbol{a}(k)$ by the equation:

$$\boldsymbol{a}(k+1) = \boldsymbol{a}(k) - r(k)\nabla l(\boldsymbol{a}(k)) \tag{4.24}$$

where r is a positive scale factor, called learning rate, which sets the step size, and as a consequence how "steep" the actual update is. The goal is this sequence of parameter vectors to finally converge to a solution minimizing $l(a)$. Thus, the number of updates (iterations) needed are problem-dependent. Another interesting issue about gradient descent methods in machine learning, is that the choice of the loss function l is based on the training set, so each step requires the use of the whole data set in order to calculate ∇l. Techniques that use the whole data set at each iteration are called batch methods. Gradient descent optimization algorithms are numerous (Adaline, Adamax, Adam, RMSprop etc), each one inserting a small variation to the classical approach. One of the most widely used is Adam (Adaptive Moment Estimation), which updates the weights using estimations of the first two moments of past gradients (mean and standard deviation).

4.3 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a special type of an artificial neural network adapted to work for time series data or data that involves sequences. RNNs were based on David Rumelhart's work in 1986. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing, speech recognition, and image captioning. They are incorporated into popular applications such as Siri, voice search, and Google Translate [20].

RNNs have the concept of 'memory' that helps them store the states or information of previous inputs to generate the next output of the sequence and so the output of RNNs depend on the prior elements within the sequence. Another distinguishing characteristic of Recurrent Networks is that they share parameters across each layer of the network. While Feed-forward Networks have different weights across each node, RNNs share the same weight parameter within each layer of the network. These weights are still adjusted in the processes of backpropagation and gradient descent to facilitate reinforcement learning [20].

An RNN's self-connections cause neuron activities (the RNN's state) to reverberate as time passes. In machine learning we typically discretize the state changes, computing them using the activities at the previous discrete

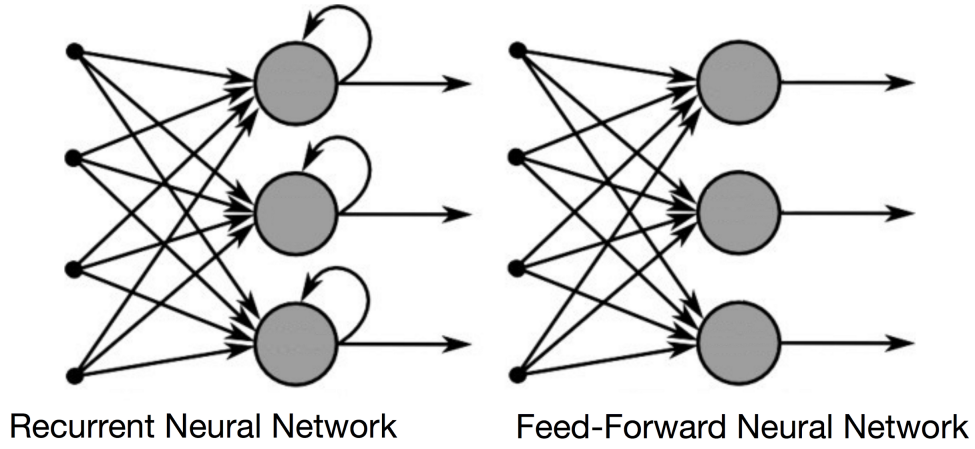


Figure 4.4: Recurrent Neural Network vs Feed-Forward Neural Network Source: A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks (Fig. 2 in [17]).

time step and the synaptic weights. This is known as "unrolling" the network [38].

We present a simplified model of RNN without bias parameters, whose activation function in the hidden layer uses the identity mapping ($\phi(x) = x$). For time step t, let the single example input and the label be $\boldsymbol{x}_t \in \mathbb{R}^d$ and y_t, respectively [68]. The hidden state $\boldsymbol{h}_t \in \mathbb{R}^h$ and the output $\boldsymbol{o}_t \in \mathbb{R}^q$ are computed as:

$$\boldsymbol{h}_t = \boldsymbol{W}_{hx}\boldsymbol{x}_t + \boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} \tag{4.25}$$

$$\boldsymbol{o}_t = \boldsymbol{W}_{qh}\boldsymbol{h}_t \tag{4.26}$$

where $\boldsymbol{W}_{hx} \in \mathbb{R}^{h\times d}, \boldsymbol{W}_{hh} \in \mathbb{R}^{h\times h}$ and $\boldsymbol{W}_{qh} \in \mathbb{R}^{q\times h}$ are the weight parameters.

We denote by $l(\boldsymbol{o}_t, y_t)$ the loss at time step t. Our objective function, the loss over T time steps from the beginning of the sequence is thus:

$$L = \frac{1}{T}\sum_{t=1}^{T} l(\boldsymbol{o}_t, y_t) \tag{4.27}$$

4.3.1 Backpropagation Through Time (BPTT)

Recurrent Neural Networks leverage Backpropagation Through Time (BPTT) algorithm which is slightly different from traditional backpropagation as it is specific to sequence data. The pronciples of BPTT are the same as traditional backpropagation, where the model trains itself by calculating errors from its output layer to its input layer.

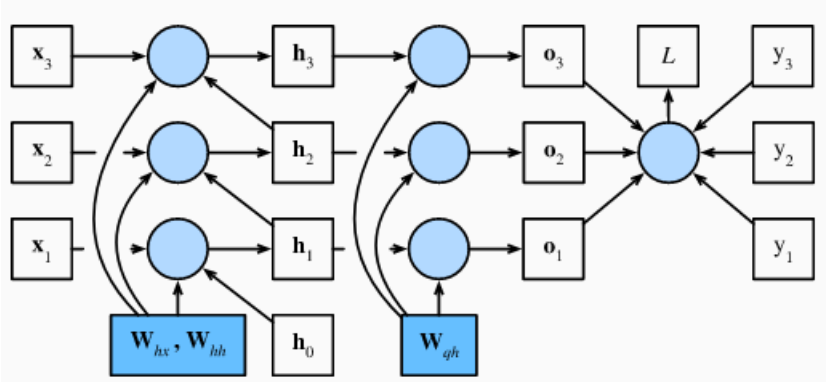


Figure 4.5: Computational graph showing dependencies for an RNN model with 3 time steps. Boxes represent variables (not shaded) or parameters (shaded) and circles represent operators. Source: Backpropagation Through Time (Fig. 2 in [68]).

The goal of BPTT is to compute the partial derivatives of the error with respect to the synaptic weights, known as the gradients. The dependencies among model variables and parameters during computation of the RNN are shown in Figure 4.5. In order to calculate and store the gradients in turn the arrows can be traversed in the opposite direction [68]. To flexibly express the multiplication of matrices, vectors and scalars of different shapes in the chain rule, we use the prod operator to multiply its arguments after the necessary operations, such as transposition and swapping input positions, have been carried out. For vectors, this is straightforward: it is simply matrix-matrix multiplication. For higher dimensional tensors, we use the appropriate counterpart. The operator prod hides all the notation overhead.

First of all, differentiating the objective function with respect to the model output at any time step t is fairly straightforward:

$$\frac{\partial L}{\partial \boldsymbol{o}_t} = \frac{\partial l(\boldsymbol{o}_t, y_t)}{T \cdot \partial \boldsymbol{o}_t} \in \mathbb{R}^q \tag{4.28}$$

The gradient of the objective function with respect to the parameter $\boldsymbol{W}_{qh}$ in the output layer is $\partial L / \partial \boldsymbol{W}_{qh}$. The objective function L depends on W_{qh}

via $o_1, ..., o_T$. Using the chain rule $\partial L/\partial \boldsymbol{W}_{qh}$ is calculated as follows:

$$\frac{\partial L}{\partial \boldsymbol{W}_{qh}} = \sum_{t=1}^{T} \text{prod}(\frac{\partial L}{\partial \boldsymbol{o}_t}, \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{W}_{qh}}) = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{o}_t} \boldsymbol{h}_t^\top \tag{4.29}$$

At the final time step T the objective function L depends on the hidden state h_T only via o_T. Therefore, the gradient $\partial L/\partial \boldsymbol{h}_T \in \mathbb{R}^h$ using the chain rule is described as:

$$\frac{\partial L}{\partial \boldsymbol{h}_T} = \text{prod}(\frac{\partial L}{\partial \boldsymbol{o}_T}, \frac{\partial \boldsymbol{o}_T}{\partial \boldsymbol{h}_T}) = \boldsymbol{W}_{qh}^\top \frac{\partial L}{\partial \boldsymbol{o}_T} \tag{4.30}$$

It gets trickier for any time step $t < T$, where the objective function L depends on h_t via h_{t+1} and o_t. According to the chain rule, the gradient of the hidden state $\partial L/\partial \boldsymbol{h}_t \in \mathbb{R}^h$ at any time step $t < T$ can be recurrently computed as:

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \text{prod}(\frac{\partial L}{\partial \boldsymbol{h}_{t+1}}, \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t}) + \text{prod}(\frac{\partial L}{\partial \boldsymbol{o}_t}, \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{h}_t}) = \boldsymbol{W}_{hh}^\top \frac{\partial L}{\partial \boldsymbol{h}_{t+1}} + \boldsymbol{W}_{qh}^\top \frac{\partial L}{\partial \boldsymbol{o}_t} \tag{4.31}$$

For analysis, expanding the recurrent computation for any time step $1 \leq t \leq T$ gives

$$\frac{\partial L}{\partial \boldsymbol{h}_t} = \sum_{i=t}^{T} (\boldsymbol{W}_{hh}^\top)^{T-i} \boldsymbol{W}_{qh}^\top \frac{\partial L}{\partial \boldsymbol{o}_{T+t-i}} \tag{4.32}$$

Finally, the objective function L depends on model parameters $\boldsymbol{W}_{hx}$ and $\boldsymbol{W}_{hh}$ in the hidden layer via hidden states $\boldsymbol{h}_1, ..., \boldsymbol{h}_T$. To compute gradients with respect to such parameters $\partial L/\partial \boldsymbol{W}_{hx} \in \mathbb{R}^{h \times d}$ and $\partial L/\partial \boldsymbol{W}_{hh} \in \mathbb{R}^{h \times h}$, we apply the chain rule that gives:

$$\frac{\partial L}{\partial \boldsymbol{W}_{hx}} = \sum_{t=1}^{T} \text{prod}(\frac{\partial L}{\partial \boldsymbol{h}_t}, \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_{hx}}) = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{h}_t} \boldsymbol{x}_t^\top \tag{4.33}$$

$$\frac{\partial L}{\partial \boldsymbol{W}_{hh}} = \sum_{t=1}^{T} \text{prod}(\frac{\partial L}{\partial \boldsymbol{h}_t}, \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{W}_{hh}}) = \sum_{t=1}^{T} \frac{\partial L}{\partial \boldsymbol{h}_t} \boldsymbol{h}_{t-1}^\top \tag{4.34}$$

BPTT can be computationally expensive as the number of timesteps increases. If input sequences are comprised of thousands of timesteps, then this will be the number of derivatives required for a single update weight update. Through this process, RNNs tend to run into two problems, known as exploding gradients and vanishing gradients. These issues are defined by

the size of the gradient, which is the slope of the loss function along the error curve. When the gradient is too small, it continues to become smaller, updating the weight parameters until they become insignificant. When that occurs, the algorithm is no longer learning. Exploding gradients occur when the gradient is too large, creating an unstable model. In this case, the model weights will grow too large, and they will eventually be represented as NaN [20].

4.3.2 Types of RNNs

There are different types of recurrent neural networks with varying architectures. While feed-forward neural networks map one input to one output, RNNs can map one to many, many to many (translation) and many to one (classifying voice) [17].

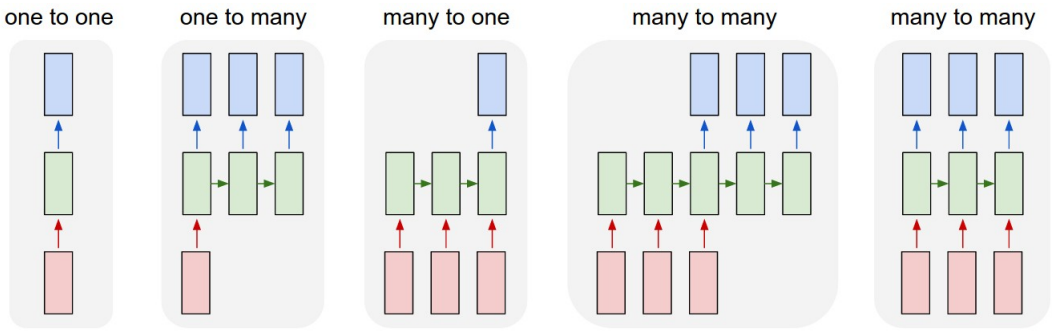


Figure 4.6: Different Types of RNN Source: The Unreasonable Effectiveness of Recurrent Neural Networks (Fig. 2 in [36]).

4.3.3 Different RNN Architectures

There are different variations of RNNs that are being applied practically in machine learning problems.

4.3.3.1 Long Short-Term Memory (LSTM)

This is a popular RNN architecture, which was introduced by Sepp Hochreiter and Juergen Schmidhuber in 1997 as a solution to vanishing gradient problem. The main work aims to address the problem of long-term dependencies. That is, if the previous state that is influencing the current prediction is not in the recent past, the RNN model may not be able to accurately predict the current state. As an example, if we wanted to predict

the italicized words in following, "Alice is allergic to nuts. She can't eat peanut butter.", the context of a nut allergy can help us anticipate that the food that cannot be eaten contains nuts. However, if that context was a few sentences prior, then it would make it difficult, or even impossible, for the RNN to connect the information [30].

To remedy this, LSTMs have "cells" in the hidden layers of the neural network, which have three gates–an input gate, an output gate, and a forget gate. These gates control the flow of information which is needed to predict the output in the network. For example, if gender pronouns, such as "she", was repeated multiple times in prior sentences, that may be excluded from the cell state [30].

4.3.3.2 Gated Recurrent Units (GRUs)

This RNN variant is similar the LSTMs as it also works to address the short-term memory problem of RNN models. It was first introduced in 2014 by Kyunghyun Cho et al. Instead of using a "cell state" regulate information, it uses hidden states, and instead of three gates, it has two—a reset gate and an update gate. Similar to the gates within LSTMs, the reset and update gates control how much and which information to retain [12].

4.3.3.3 Bidirectional Recurrent Neural Networks (BRNN)

BRNN was invented in 1997 by Schuster and Paliwal so as to increase the amount of input information available to the network. While unidirectional RNNs can only drawn from previous inputs to make predictions about the current state, bidirectional RNNs pull in future data to improve the accuracy of it. The principle of BRNN is to split the neurons of a regular RNN into

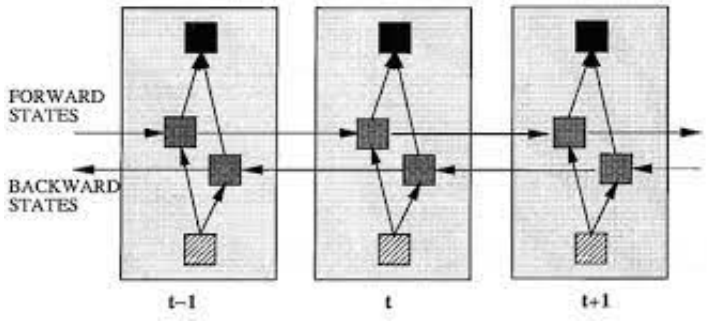


Figure 4.7: General structure of the bidirectional recurrent neural network (BRNN) shown unfolded in time for three time steps. Source: Bidirectional Recurrent Neural Networks (Fig. 1 in [51]).

two directions, one for positive time direction (forward states), and another for negative time direction (backward states). Outputs from forward states are not connected to inputs of backward states, and vice versa. This leads to the general structure that can be seen in Figure 4.7, where it is unfolded over three time steps. Note that without the backward states, this structure simplifies to a regular unidirectional forward RNN, as shown in Fig. 1. If the forward states are taken out, a regular RNN with a reversed time axis results. With both time directions taken care of in the same network, input information in the past and the future of the currently evaluated time frame can directly be used to minimize the objective function without the need for delays to include future information, as for the regular unidirectional RNN discussed above [51].

The BRNN can principally be trained with the same algorithms as a regular unidirectional RNN because there are no interactions between the two types of state neurons and, therefore, can be unfolded into a general feed-forward network. However, if, for example, any form of back-propagation through time (BPTT) is used, the forward and backward pass procedure is slightly more complicated because the update of state and output neurons can no longer be done one at a time. For forward pass, forward states and backward states are passed first, then output neurons are passed. For backward pass, output neurons are passed first, then forward states and backward states are passed next. After forward and backward passes are done, the weights are updated [51].

4.3.3.4 Sequence to Sequence Model

A Sequence to Sequence (Seq2Seq) model consists of two Recurrent Neural Networks. The most common architecture used to build Seq2Seq models is Encoder-Decoder architecture. The encoder and decoder work simultaneously either using the same parameter or different ones. Both encoder and the decoder are LSTM models (or sometimes GRU models) [58].

Encoder reads the input sequence and summarizes the information in something called the internal state vectors or context vector (in case of LSTM these are called the hidden state and cell state vectors). The outputs of the encoder are discarded and only the internal states are preserved [2].

The decoder is an LSTM whose initial states are initialized to the final states of the Encoder LSTM, i.e. the context vector of the encoder's final cell is input to the first cell of the decoder network. Using these initial states, the decoder starts generating the output sequence, and these outputs are also taken into consideration for future outputs. This context vector aims to encapsulate the information for all input elements in order to help the

decoder make accurate predictions [2].

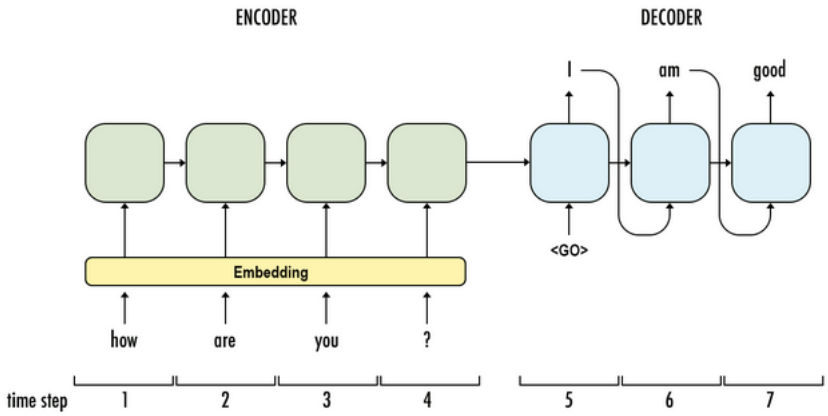


Figure 4.8: Representation of the architecture a Sequence to Sequence Model in a specific example. Source: Types of Neural Networks and Definition of Neural Network(Fig. 12 in [58]).

This model, on contrary to the actual RNN, is particularly applicable in those cases where the length of the input data is equal to the length of the output data. While they possess similar benefits and limitations of the RNN, these models are usually applied mainly in chatbots, machine translations, and question answering systems [58].

Chapter 5

Vector Representation of Language

Before referring to others models for Natural Language Processing, it is important to analyze how computers process natural language. Natural language is one of the most complex tools used by humans for a wide range of reasons, for instance to communicate with others, to express thoughts, feelings, and ideas, to ask questions, or to give instructions. Therefore, it is crucial for computers to possess the ability to use the same tool in order to effectively interact with humans. In fact, computers represent language as vectors.

In the following sections we discuss several methods for creating such a vector representation of language and the theory behind them.

5.1 One-hot Representation

The simplest form of word representation is one-hot encoding, which established the basis of word vector space models. Assume we have 100 words in a vocabulary and we would like to encode them as one-hot representations. First, we associate an index (between 1 to 100) to each word. Then, each word is represented as a 100-dimension array-like representation, in which all the dimensions are zero except for the one corresponding to its index, which is set to one.

However, simple one hot vectors are not a very useful input to most natural language processing tasks, because they are embedded in a vector space that does not contain any extra meaning information about the words being represented. Each word is assigned a different representation and there is no notion of "similarity" between them. For example, using this representation, it is not possible to encode the conceptual similarity between "noon" and "midday". Even worse, the two similar looking words such "desk"

and "desks" (which would have similar string-based representations) are assigned completely different one-hot vectors. Moreover, the dimensionality of one-hot representations grows with the number of words in the vocabulary. In a typical vocabulary, we should expect hundreds of thousands of words. Representing each word using one-hot representation is definitely too storage-intensive and would make the processing difficult [46].

5.2 Vector Space Models

The Vector Space Model (VSM), first proposed by Salton et al. [49], provides a more flexible solution to the limitations of one-hot representation. In this model, objects are represented as vectors in an imaginary multi-dimensional continuous space. In NLP, the space is usually referred to as the semantic space and the representation of the objects is called distributed representation. Objects can be words, documents, sentences, concepts, or entities, or any other semantic carrying item between which we can define a notion of similarity [46].

The representations that are generated using neural networks are commonly referred to as embedding, particularly due to their property of being dense and low dimensional. We can call these representations semantic embeddings. Semantic embedding refers to a series of representation learning (or feature learning) techniques that encode the semantics of data such as sequences and graphs into vectors, such that they can be utilized by downstream machine learning prediction and statistical analysis tasks.

Sequence feature learning models such as Feed-Forward Neural Networks and Recurrent Neural Networks, that were described in the previous chapter (Chapter 3), are widely used for semantic embedding [11]. In this section, we will talk about the foundations behind constructing semantic spaces, particularly for words.

5.2.1 Word Embeddings

Semantic spaces are constructed automatically by analyzing word co-occurrences in large text corpora. Words that occur in similar contexts tend to have similar meanings. This link between similarity in how words are distributed and similarity in what they mean is called the distributional hypothesis. The hypothesis was first formulated in the 1950s by linguists Joos [33], Harris [29] and Firth [22], who noticed that words which are synonyms (like oculist and eye-doctor) tended to occur in the same environment (e.g., near words like eye or examined) with the amount of meaning difference

between two words "corresponding roughly to the amount of difference in their environments" [14].

Word embeddings are in fact a special type of distributed word representation that are constructed by leveraging neural networks, mainly popularised after 2013, with the introduction of Word2vec. Word embeddings are usually classified as predictive models because they are computed through language modeling objectives, such as predicting the next or a missing word [46].

5.2.2 Predictive Models

In the last decade, together with the growth of deep learning, embeddings have dominated the field and predictive models have replaced the conventional count-based models. Two of the most popular word embedding models are Word2vec and GloVe.

5.2.2.1 Word2vec

Word2vec is a well known group of sequence feature learning techniques for learning word embeddings from a large corpus, and was initially developed by a team at Google in 2013 [41]. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, Word2vec represents each distinct word with a particular list of numbers called a vector. The vectors are chosen carefully such that a simple mathematical function indicates the level of semantic similarity between the words represented by those vectors.

Word2vec is based on a simple but efficient feedforward neural architecture which is trained with language modeling objective. Word2vec can be configured to use either of two classic auto-encoding architectures for learning representations of sequential items: continuous Skip-gram and continuous Bag-of-Words (CBOW).

The CBOW model aims at predicting the current word using its surrounding context. The Skip-gram model is similar to the CBOW model but in this case the goal is to predict the words in the surrounding context given the target word, rather than predicting the target word itself. A simplification of the general architecture of the CBOW and Skip-gram models of Word2vec is represented in Figure 5.1. The architecture consists of input, projection (hidden) and output layers. The input layer has the size of the word vocabulary and encodes the context as a combination of one-hot vector representations of surrounding words of a given target word. The output layer has the same size as the input layer and contains a one-hot vector of the target word during the training phase [41].

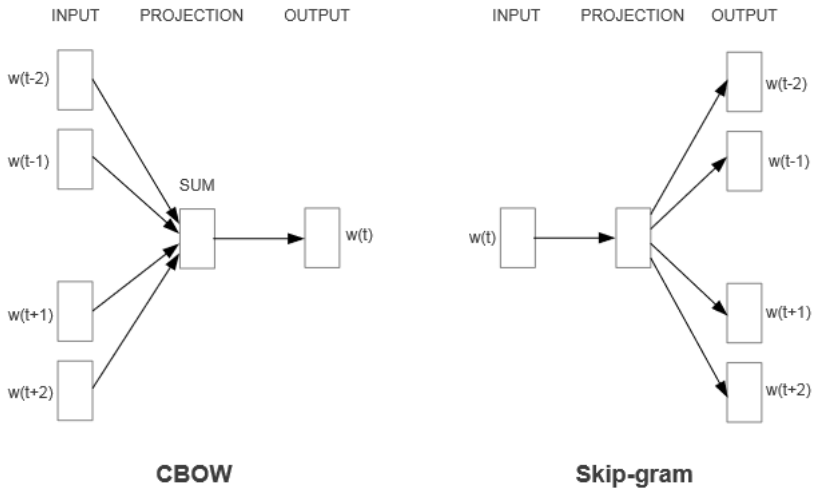


Figure 5.1: Learning architecture of the CBOW and Skip-gram models of Word2vec Source: Efficient Estimation of Word Representations in Vector Space (Fig. 1 in [41]).

5.2.2.2 GloVe

GloVe is developed as an open-source project at Stanford and was first launched in 2014. GloVe is a model for distributed word representation. The model is an unsupervised learning algorithm for obtaining vector representations for words. This is achieved by mapping words into a meaningful space where the distance between words is related to semantic similarity. Training is performed on aggregated global word-word co-occurrence statistics from a corpus [45].

5.2.3 Similarity Metrics

In order to quantify the similarity between two embeddings, we need to present some similarity metrics. The two most popular metrics for calculating similarity are Euclidean Distance and Cosine Similarity. We consider two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ of dimension n.

- **Euclidean Distance**

$$d(\boldsymbol{u}, \boldsymbol{v}) = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2} \tag{5.1}$$

where u_i and v_i are components of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ respectively.

- **Cosine Similarity**

$$sim(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u} \cdot \boldsymbol{v}}{\|\boldsymbol{u}\|\|\boldsymbol{v}\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2}\sqrt{\sum_{i=1}^{n} v_i^2}} \tag{5.2}$$

  where u_i and v_i are components of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ respectively.

  The resulting similarity ranges from -1 meaning exactly opposite, to 1 meaning exactly the same, with 0 indicating orthogonality or decorrelation, while in-between values indicate intermediate similarity or dissimilarity.

5.2.4 Contextualized Embeddings

The problem of the pretrained word embeddings, such as Word2vec and GloVe, is that they compute a single static representation for each word. The representation is fixed and independent from the context in which the word appears. Therefore, the static representation of words can substantially hamper the ability of NLP systems to understand the semantics of the input text.

Unlike static word embeddings, contextualized embeddings are representations of words in context. These embeddings are dynamic and the same word can be assigned different embeddings if it appears in different contexts. Instead of receiving words as distinct units and providing independent word embeddings for each, contextualized models receive the whole text span (the target word along with its context) and provide specialized embeddings for individual words which are adjusted to their context.

These context-sensitive embeddings are in fact the internal states of a deep neural network which is trained with language modeling objectives either in an unsupervised manner or on a supervised task. The training of contextualized embeddings is carried out at a pretraining stage, independently from the main task, on a large unlabeled (or differently labeled) text corpus. The trained model can then generate contextualised representations for all the words in the given text.

RNNs, which were represented in the previous chapter (Section 4.3), and mostly LSTM constitute a good contextualized representation model. However, most of the recent literature on contextualized embeddings is based on a novel model called Transformer. Today, Transformers are dominantly exceeding the performance levels of conventional recurrent models on most NLP tasks that involve sequence encoding. So, it is important to provide a brief overview of Transformer on Chapter 6.

Chapter 6

The Transformer Model

As we described on Chapter 4, a popular approach for language modeling is RNNs as they capture dependencies between words well, especially when using modules such as LSTM. However, RNNs tend to be slow and their ability to learn long-term dependencies is still limited due to vanishing gradients.

In 2017 a supervised deep learning model, which was called Transformer, was first introduced by a team at Google Brain and is increasingly the model of choice for NLP problems, replacing RNN models, such as LSTM and GRUs with added attention mechanisms. This feature allows for more parallelization than RNNs and therefore reduces training times. The original Transformer network was trained on translation tasks. Sentences were translated to German from English and to French from English [60].

Like RNNs, Transformers are designed to handle sequential input data, such as natural language, for tasks such as translation and text summarization. However, unlike RNNs, Transformers do not necessarily process the data in order. Rather, the attention mechanism provides context for any position in the input sequence. For example, if the input data is a natural language sentence, the Transformer does not need to process the beginning of the sentence before the end. Rather, it identifies the context that confers meaning to each word in the sentence [66].

The Transformer is the first transduction model relying entirely on self-attention to compute representation of its input and output without using sequence-aligned RNNs or convolution, highlighting the fact that attention mechanisms alone can match the performance of RNNs with attention. Over time, the Transformer architecture has become an effective and efficient replacement to RNN-based models in a variety of domains involving sequential data such as natural language processing, speech, and video-related tasks. It is a precursor to some of the most popular natural language processing models such as BERT and GPT [66].

In the following sections we describe Transformer's Model Architecture (Section 6.1) and then we introduce BERT model (Section 6.2) and four other Transformer models (Section 6.3), such as RoBERTa (Section 6.3.1), DistilBERT (Section 6.3.2), Longformer (Section 6.3.3) and BioBERT (Section 6.3.4).

6.1 Model Architecture

The Transformer network consists of an encoding as well as a decoding component. The encoder maps an input sequence of symbol representations $(x_1, x_2, ..., x_n)$ to a sequence of continuous representations $z = (z_1, z_2, ..., z_n)$. Given z, the decoder then generates an output sequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive [26], consuming the previously generated symbols as additional input when generating the next. The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder [60].

6.1.1 Encoder - Decoder

The encoder is a feed forward network consisting of a stack of $N = 6$ identical layers, each composed of two sub-layers. The first sub-layer is an attention layer known as the multi-head self-attention layer. The second sub-layer is a position-wise feed forward network (FFN) layer. Each sub-layer has residual connections around it, followed by layer-normalization [60].

The multi-head self-attention layer is composed of several parallel layers known as self-attention layers. The self-attention mechanism relates input tokens and their positions within the same input sequence. Such parallel stacking of several self-attention layers achieves more expressiveness as opposed to a single attention formulation. The particular form of attention used in the Transformer is known as the scaled dot-product attention [60].

The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, there are residual connections around each of the sub-layers, followed by layer normalization [60].

The decoder also uses restrictions to ensure that the prediction for a particular position in the sequence depends only on the previous elements of the sequence and not the subsequent ones. This is achieved by offsetting the

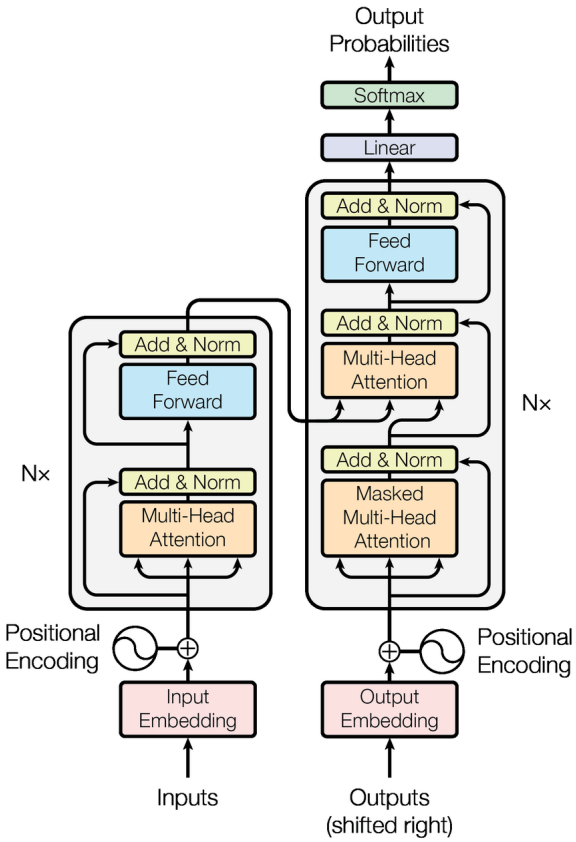


Figure 6.1: The Transformer Model Architecture. Source: Attention Is All You Need (Fig. 1 in [60]).

output embeddings by one position and by masking the self-attention in the decoder to prevent it from using knowledge of subsequent positions [60].

6.1.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key [60].

6.1.2.1 Scaled Dot-Product Attention

In the case of the Transformer, the specialized attention function is known as scaled dot-product attention, which is a scaled version of the dot-product

attention. The only difference with the dot-product attention is the division by the scaling factor $\sqrt{d_k}$, where d_k is the dimension of the keys. The input consists of queries and keys of dimension d_k, and values of dimension d_v. The dot products of the query with all keys are each divided by $\sqrt{(d_k)}$ and then a softmax function is applied in order to obtain the weights on the values. In practice, the attention function is computed on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V [60]. The matrix of output is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{6.1}$$

The scaling term prohibits the dot product from being affected by keys of large dimensions, which may lead the softmax functions into regions of extremely small gradients.

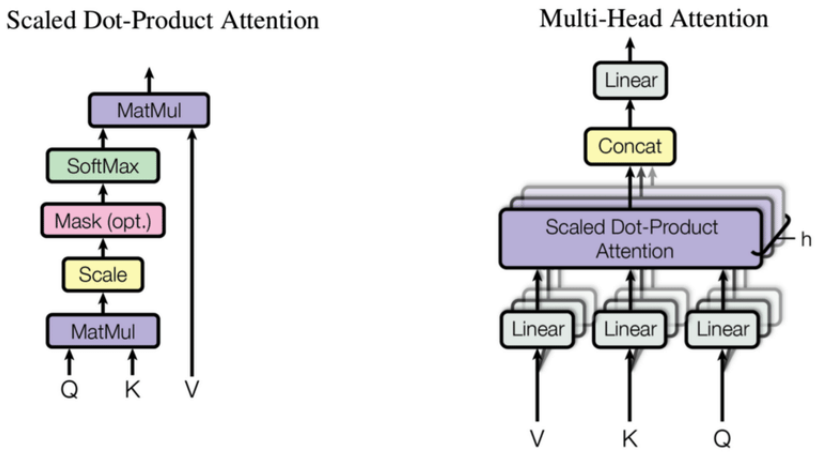


Figure 6.2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. Source: Attention Is All You Need (Fig. 2 in [60]).

6.1.2.2 Multi-Head Attention

Instead of performing a single attention function, attention from multiple perspectives may allow the model to jointly utilize information from different representation subspaces at different positions. A multi-head attention block runs several attention functions in parallel on linear projections of the same

queries, keys, and values. It then concatenates the results and further projects them to arrive at a single output, just like regular attention would [60].

For creating a single head, each of the inputs is projected by multiplying with a parameter matrix. The formulation for the h_{th} head is just the attention function applied to the h_{th} linear projections of the queries, keys and values. With all heads computed in parallel, they are first concatenated and then multiplied with a multi-head output parameter matrix W^O to arrive at the multi-head formulation. The Transformer uses 8 heads in the multi-head attention [60].

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(head_1, ..., head_h)W^O \\ \text{where} \quad head_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{6.2}$$

6.1.2.3 Self Attention

Self-attention is a mechanism which allows the model to relate different positions in the input sequence to each other. For example, this enables the network to understand what the word “he” refers to in the tokenized sentence “my dad baked a cake because he was happy”. The multi-head attention is used in three different ways in the Transformer [60]. In "encoder-decoder attention" layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence [60]. The encoder contains self-attention layers. In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder [60]. The decoder also has several identical layers and there is a multi-head attention block between adjoining layers. Unlike the encoder, the decoder is prohibited from looking at subsequent positions of the output. It can only utilize queries, keys and values from the previous positions. Therefore, the self-attention in the case of the decoder is known as a masked multi-head attention. The masking merely hides information of subsequent positions from visibility to a decoder position. With this mask in place, for decoder self-attention, the inputs are sourced in the same way as that of encoder self-attention [60].

6.1.3 Position-Wise Feed-Forward Network

Each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identi-

cally. This consists of two linear transformations with a ReLU activation in between [60].

Let W_1, W_2, b_1, b_2 be the weights and biases of the first (hidden) and second (output) level FFN and x the input. The output of FFN is:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{6.3}$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer. Another way of describing this is as two convolutions with kernel size 1.

6.1.4 Positional Encoding

Since the model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, "positional encodings" are added to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension d_{model} as the embeddings, so that the two can be summed. In the original Transformer, for position *pos* and dimension i of the input embedding, the positional encoding PE is computed as [60]:

$$PE_(pos, 2i) = sin(pos/10000^{2i/d_{model}}) \tag{6.4}$$

6.2 Bi-Directional Encoder Representation From Transformers (BERT)

BERT (Bidirectional Encoder Representations from Transformers) is a language representation language model released in 2018 by researchers at Google AI Language. BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications [16].

6.2.1 Input and Output Representations

The input to the lowest layer in the BERT network is able to unambiguously represent both a single sentence and a pair of sentences in one token sequence. A “sentence” can be an arbitrary span of contiguous text, rather

than an actual linguistic sentence while a "sequence" refers to the input token sequence to BERT, which may be a single sentence or two sentences packed together. WordPiece embeddings, which includes a 30000 token vocabulary, is used to create tokens. The max sequence length is 512 tokens [16].

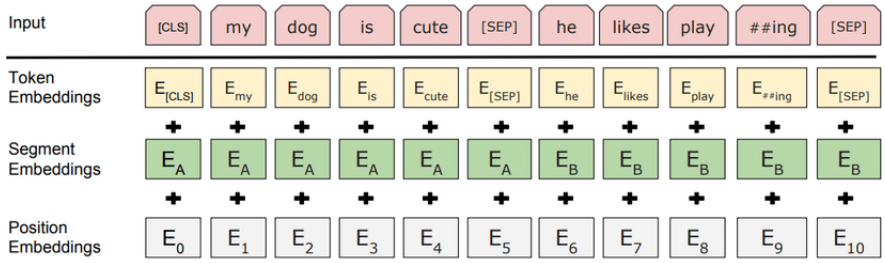


Figure 6.3: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings. Source: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Fig. 2 in [16]).

The first token of every sequence is always a special classification token ([CLS]) while the last token of each sentence is a special token (SEP). Similar to the Transformer, BERT adds to the token segment embeddings to inform the model of which sentence in a sentence pair the token belongs and positional embeddings to inform the model of where in the sequence the token belongs. To summarize, each input embedding consists of the summation of a token embedding, a positional embedding and a segment embedding [16]. A visualization of this construction can be seen in Figure 6.3.

6.2.2 Pre-training BERT

The BERT network is trained using two different unsupervised pre-training tasks, which enable the bidirectional representation of each token and enable the model to understand the relationship between two sentences. The pre-training procedure largely follows the existing literature on language model pre-training and for the pre-training corpus the BooksCorpus and English Wikipedia are used [16].

6.2.2.1 Masked Language Model (MLM)

BERT uses a masked language procedure to enable bidirectional representations. This allows the model to condition on both the forward as well as the backward context. In order to train a deep bidirectional representation, 15%

of the input tokens are masked at random using a special [MASK] token and then these masked tokens are predicted. This procedure is called "Masked LM", but it is often referred as "Cloze task" [57]. In this case, the final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, As loss function, cross-entropy is used [16].

Since the [MASK] token does not appear during fine-tuning, a downside is that a mismatch between pre-training and fine-tuning is created. To mitigate this, the tokens randomly chosen to be masked are not always masked. 80% of the time, the randomly chosen token is replaced by [MASK] token, 10% of the time the token is replaced by a randomly chosen token from the vocabulary and the rest of the time the token remains unchanged. Using this procedure, the encoder does not know which token it has to predict, forcing the model to keep a representation of each token which depends on the surrounding context [16].

Assuming the unlabeled sentence is "my dog is hairy" and during the random masking procedure the 4-th token is chose (which corresponding to hairy), the masking procedure can be further illustrated by:

- **80% of the time:** Replace the word with the [MASK] token, e.g., my dog is hairy → my dog is [MASK].
- **10% of the time:** Replace the word with a random word, e.g., my dog is hairy → my dog is apple.
- **10% of the time::** Keep the word unchanged, e.g., my dog is hairy → my dog is hairy. The purpose of this is to bias the representation towards the actual observed word.

The advantage of this procedure is that the Transformer encoder does not know which words it will be asked to predict or which have been replaced by random words, so it is forced to keep a distributional contextual representation of every input token. Additionally, because random replacement only occurs for 1.5% of all tokens (i.e., 10% of 15%), this does not seem to harm the model's language understanding capability [16].

6.2.2.2 Next Sentence Prediction (NSP)

Next sentence prediction is important for tasks that are based on understanding the relationship between two sentences, such as question answering. In order to train a model that understands sentence relationships, BERT is pre-trained for a binarized next sentence prediction task. More specifically, when choosing the sentences A and B for each pre-training example, 50% of

the time B is the actual next sentence that follows A (labeled as IsNext) and 50% of the time it is a random sentence from the corpus (labeled as NotNext) [16].

6.2.3 Mean Pooling Operation

The token embeddings outputted by BERT can be used to create a single vector encoding, sentence embeddings. To produce a vector like this the mean pooling operation is used where the individual token embeddings excluding the non-real tokens ([CLS] and [SEP]) are averaged. Max pooling operation, also, exists in which the maximum of the token embeddings is taken. Since BERT consists of multiple stacked Transformer encoders, the token embeddings can be excluded from any layer. The output embedding length is 768 [16].

6.2.4 Fine-tuning BERT

BERT can also be fine-tuned for a multitude of tasks. In the fine-tuning training, most hyper-parameters stay the same as in BERT training, and the BERT team gives specific guidance on the hyper-parameters that require tuning. The BERT team has used this technique to achieve state-of-the-art results on a wide variety of challenging natural language tasks [16].

6.3 Other Transformer Models

Google's BERT and recent transformer-based methods have taken the NLP landscape by a storm, outperforming the state-of-the-art on several tasks. Over time many new models have been inspired by the BERT architecture but are trained in different languages or optimized on domain-specific data sets. There is continuous progress happening and many optimized versions are introduced often [34]. In this section, some of the off-the-shelf pre-trained models of BERT are presented.

6.3.1 RoBERTa

Known as a 'Robustly Optimized BERT Pretraining Approach' RoBERTa is a BERT variant developed to enhance the training phase. RoBERTa was developed by training the BERT model longer, on larger data of longer sequences and large mini-batches. Facebook researchers, who introduced RoBERTa, obtained substantially improved results with some modifications of BERT hyperparameters [40].

6.3.2 DistilBERT

DistilBERT learns a distilled (approximate) version of BERT, retaining 97% performance but using only half the number of parameters. Specifically, it does not has token-type embeddings, pooler and retains only half of the layers from Google's BERT. DistilBERT uses a technique called distillation, which approximates the Google's BERT, i.e. the large neural network by a smaller one. The idea is that once a large neural network has been trained, its full output distributions can be approximated using a smaller network. This is in some sense similar to posterior approximation. One of the key optimization functions used for posterior approximation in Bayesian Statistics is Kulback Leiber divergence and has naturally been used here as well [50].

6.3.3 Longformer

Transformer-based models are unable to process long sequences due to their self-attention operation, which scales quadratically with the sequence length. To address this limitation, the Longformer was introduced with an attention mechanism that scales linearly with sequence length, making it easy to process documents of thousands of tokens or longer. Longformer's attention mechanism is a drop-in replacement for the standard self-attention and combines a local windowed attention with a task motivated global attention [6].

6.3.4 BioBERT

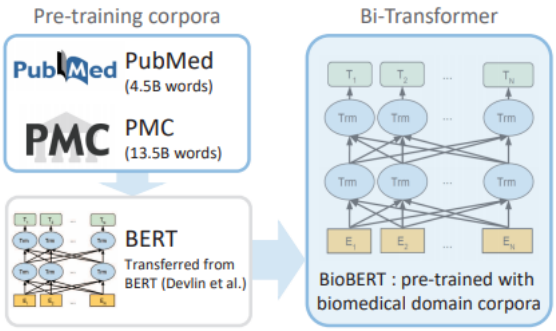


Figure 6.4: Overview of the pre-training BioBERT. Source: BioBERT: a pre-trained biomedical language representation model for biomedical text mining (Fig. 1 in [37].

Nowadays, biomedical text mining is becoming increasingly important as the number of biomedical documents rapidly grows. With the progress in

NLP. Extracting valuable information from biomedical literature has gained popularity among researchers, and deep learning has boosted the development of effective biomedical text mining models. However, biomedical domain texts contain a considerable number of domain-specific proper nouns and terms, which are understood mostly by biomedical researchers. As a result, NLP models designed for general purpose language understanding often obtains poor performance in biomedical text mining tasks.

BioBERT, which stands for Bidirectional Encoder Representations from Transformers for Biomedical Mining, is a variation of BERT model and it was introduced in 2019 by a research team from Korea University and Clova AI. Researchers pre-trained BERT on biomedical corpora, specifically on PubMed abstracts (PubMed) and PubMed Central full-text articles (PMC). By having a pre-trained model that encompasses both general and biomedical domain corpora, developers and practitioners could now encapsulate biomedical terms that would have been incredibly difficult for a general language model to comprehend [37].

Chapter 7

Knowledge Representation

Knowledge representation and reasoning is the field of artificial intelligence (AI) dedicated to represent information about the world in a form that a computer system can use. It actually incorporates findings from psychology about how humans solve problems and represent knowledge in order to design formalisms that will make complex systems easier to design and build, especially expert systems. Knowledge representation and reasoning also incorporates findings from logic to automate various kinds of reasoning, such as the application of rules or the relations of sets and subsets. The knowledge representation formalisms include ontologies and knowledge graphs.

Ontology was defined in 1993 by Gruber and in 1997 this definition was adjusted more appropriately by Borst. In 1998, Studer and others adjusted the two definitions in the following definition proposal: "An ontology is a formal, explicit specification of a shared conceptualization" [28]. Several authors have refined the definitions over time to indicate, more clearly, that an ontology is a formally-defined vocabulary for a particular domain of interest used to capture knowledge about that (restricted) domain of interest. So, an ontology describes the concepts in the domain and also the relationships that hold between those concepts [9].

There are a number of such languages for ontologies, both proprietary and standards-based. The Web Ontology Language (OWL) is one of the most famous family of knowledge representation languages for authoring ontologies. The OWL languages are characterized by formal semantics. They are built upon the World Wide Web Consortium's (W3C) standard for objects called the Resource Description Framework (RDF).

There are three variants of OWL with different levels of expressiveness. OWL DL (description logic) is one of them and is designed to provide the maximum expressiveness possible while retaining computational completeness, decidability and the availability of practical reasoning algorithms. OWL DL

includes all OWL language constructs, but they can be used only under certain restrictions. Complex concepts and roles can be composed using DL constructors such as conjunction, disjunction, existential restriction and universal restrictions. An OWL ontology comprises a TBox T and an ABox A. The TBox is a set of axioms such as General Concept Inclusion (GCI) axioms, Role Inclusion (RI) axioms and Inverse Role axioms. The ABox is a set of assertions such as concept assertions, role assertions and individual equality and inequality assertions.

In OWL, the aforementioned concept, role and individual are modeled as class, object property and instance, respectively. There are, also, data properties and annotation properties. We refer to classes, properties and instances as entities. Object property models the relationship between two instances, a data property models the relationship between an instance and a literal value (number or text) and an annotation property is used to represent a (non-logical) relationship between an entity and an annotation (e.g., comment or label). Each entity is uniquely represented by an Internationalized Resource Identifier (IRI). In OWL, complex concepts, complex roles, axioms and role assertions can be serialised as (sets of) RDF triples, each of which is a tuple composed of a subject, a predicate and an object. In addition to axioms and assertions with formal logic-based semantics, an ontology often contains metadata information in the form of annotation axioms. These annotations can also be represented by RDF triples using annotation properties [11].

One the other hand, the definitions of knowledge graphs (KG) vary and there is research which suggests that a knowledge graph is no different than an ontology [21]. The term was popularized by the Google's Knowledge Graph in 2012. In fact, KG refers to structured knowledge resources which are often expressed as a set of RDF triples. Many KGs only contain instances and facts which are equivalent to an OWL ontology ABox. Some other KGs are also enhanced with a schema which is equivalent to the TBox of an OWL ontology. Thus, a KG can often be understood as an ontology [11].

The use of external knowledge sources such as ontologies and knowledge graphs can be used in order to produce embeddings for their terms with the aim of semantically enhancing a natural language processing (NLP) model. In the following sections we describe Graph Embeddings (Section 7.1) and Ontology Embeddings (Section 7.2) and some algorithms that produce them. We particularly focus on the ontology embedding algorithm OWL2Vec* (Section 7.2.1), which has been used at the experimental study of this diploma thesis.

7.1 Graph Embeddings

Semantic embeddings, that were described on Chapter 3, have also been extended to KGs composed of role assertions. The entities and relations (object properties) are represented in a vector space while retaining their relative relationships (semantics).

One technique for learning KG representations is computing the embeddings iteratively adjusting the vectors using an optimization algorithm to minimize the overall loss across all the triples. Algorithms based on this technique include translation based model such as TransE [8] and TransR [39]. Another technique is to first explore the neighborhoods of entities and relations in the graph, and then learn the embeddings using a word embedding model. One representative algorithm based on this technique is node2vec [27], which extracts random graph walks and creates skip-gram or CBOW models as the corpus for training. Another is Deep Graph Kernels [67], which uses graph kernels such as Weisfeiler-Lehman (WL) sub-graph kernels as the corpus. However, both embedding algorithms were originally developed for undirected graphs, and thus may have limited performance when directly applied to KGs. RDF2Vec [47] addresses this issue by extending the idea of the above two algorithms to directed labeled RDF graphs and has been shown to learn effective embeddings for large scale KGs.

7.2 Ontology Embeddings

In recent years, the use of machine learning prediction and statistical analysis with ontologies is receiving wider attention and as a result many ontology embedding algorithms have been developed. The objective of OWL ontology embedding is to represent each OWL named entity (class, instance or property) by a vector, such that the inter-entity relationships indicated by the above information are kept in the vector space, and the performance of the downstream tasks, where the input vectors can be understood as learned features, is maximized.

Onto2Vec [53] and OPA2Vec [54] are two ontology embedding algorithms using a model of either the skip-gram architecture or the CBOW architecture. Onto2Vec uses the axioms of an ontology as the corpus for training, while OPA2Vec complements the corpus of Onto2Vec with the lexical information provided by, e.g., $rdfs{:}comment$. Both methods treat each axiom as a sentence, which means that they cannot explore the correlation between axioms. This makes it hard to fully explore the graph structure and the logical relation between axioms, and may also lead to the problem of corpus

shortage for small to medium scale ontologies.

OWL2Vec* is another method for generating ontology embeddings, which deals with the above issues of OPA2Vec and Onto2Vec. It actually complements their axiom corpus with a corpus generated by walking over RDF graphs that are transformed from the OWL ontology with its graph structure and logical constructors considered. Furthemore, OWL2Vec* creates embeddings for not only the ontology entities as the previous KG/ontology embedding methods but also for the words in the lexical information. OWL2Vec* embedding targets OWL ontologies, which are based on the $SROIQ$ DL [11].

7.2.1 OWL2Vec*

In 2021 Jiaoyan Chen, Pan Hu, Ernesto Jimenez-Ruiz, Ole Magnus Holter, Denvar Antonyrajah and Ian Horrocks proposed a random walk and word embedding based ontology embedding method named OWL2Vec* [11], which encodes the semantics of an OWL ontology by taking into account its graph structure, lexical information and logical constructors. This team first applied this framework in three different real world datasets and showed OWL2Vec* benefits from these three different aspects of an ontology in class membership prediction and class subsumption prediction tasks.

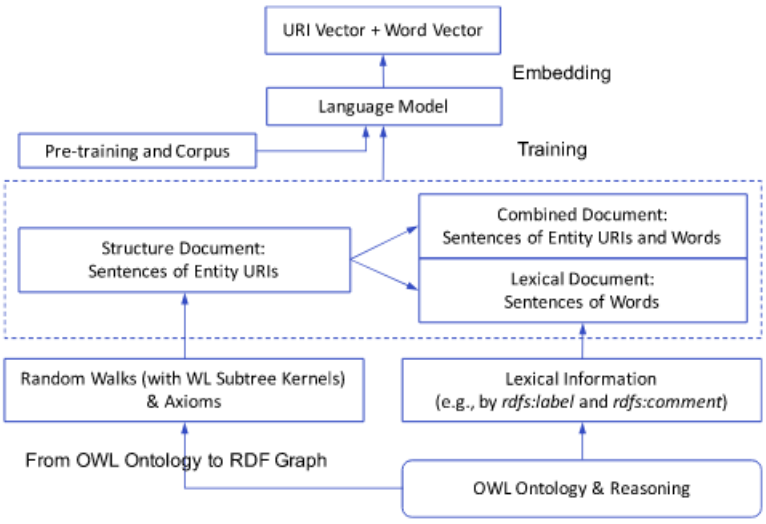


Figure 7.1: The Overall Framework of OWL2Vec*. Source: OWL2Vec*: embedding of OWL ontologies (Fig. 2 in [11]).

The overall framework of OWL2Vec* mainly consists of two core steps: (i) corpus extraction from the ontology, and (ii) language model training

Axiom of condition 1	Axiom of triple(s) of condition 2	Projected triple(s)
$A \sqsubseteq \square r.D$	$D \equiv B \vert B_1 \sqcup ... \sqcup B_n \vert B_1 \sqcap ... \sqcap B_n$	$\langle A, r, B \rangle$ or
or		
$\square r.D \sqsubseteq A$		
$\exists r.T \sqsubseteq A$ (domain)	$\top \sqsubseteq \forall r.B$ (range)	$\langle A, r, B_i \rangle$ for $i \in 1, ..., n$
$A \sqsubseteq \exists r.\{b\}$	$B(b)$	
$r \sqsubseteq r'$	$\langle A, r', B \rangle$ has been projected	
$r' \equiv r^-$	$\langle B, r', A \rangle$ has been projected	
$s_1 \circ ... \circ s_n \sqsubseteq r$	$\langle A, s_1, C_1 \rangle ... \langle C_n, s_n, B \rangle$ have been projected	
$B \sqsubseteq A$	–	$\langle B, rdfs{:}subClassOf, A \rangle$
		$\langle A, rdfs{:}subClassOf^-, B \rangle$
$A(a)$	–	$\langle a, rdf{:}type, A \rangle$
		$\langle A, rdf{:}type^-, a \rangle$
$r(a, b)$	–	$\langle a, r, b \rangle$

Table 7.1: Projection rules used in the second strategy to generate an RDF graph. $\sqsubseteq$ is one of: $\geq, \leq, =, \exists, \forall$, A, B, B_i and C_i are atomic concepts (classes), s_i, r and r' are roles (object properties), r^- is the inverse of a relation r, a and b are individuals (instances), $\top$ is the top concept (defined by $owl{:}Thing$)

with the corpus and entity embedding. The corpus includes a structure document, a lexical document, and a document combining the structure and the lexical information. The former two aim at exploring the ontology's graph structure, logical constructors and lexical information, while the third aims at preserving the correlation between entities (URIs) and their lexical labels (words). Briefly, given an input ontology O and the target entities E of O for embedding, OWL2Vec* outputs a vector for each entity e in E, denoted as $e \in \mathbb{R}^d$, where d is the (configurable) embedding dimension.

OWL2Vec* focuses on OWL ontologies instead of typical KGs, with the goal of preserving the semantics not only of the graph structure, but also of the lexical information and the logical constructors. It is worth point out that the graph of an ontology, which includes hierarchical categorization structure, differs from the multi-relation graph composed of role (relation) assertions of a typical KG. Furthermore, there are currently no existing KG embedding methods that jointly explore the ontology's lexical information and logical constructors.

From OWL ontology to RDF graph

In order to turn the original OWL ontology O into a graph G in RDF form, OWL2Vec* incorporates two strategies. The first strategy implements the transformation according to the OWL to RDF Graph Mapping defined by the W3C [1]. Some simple axioms such as membership and subsumption

[1] `https://www.w3.org/TR/owl2-mapping-to-rdf/`

axioms for atomic entities, data and annotation properties associated to atomic entities and relational assertions between atomic instances can be directly transformed into RDF triples by introducing some built-in properties or using the bespoke properties in the axioms. Axioms involving complex class expressions need to be transformed into multiple triples and often rely on blank nodes.

The second strategy is based on the projection rules proposed in Table 7.1 [55] [31]. Every RDF triple $\langle X, r, Y \rangle$ in the projection (the third column) is justified by one or more axioms in the ontology (the first and the second columns). As in the first strategy, a simple relational assertion between two atomic entities (the last row in Table 7.1), or a simple data or annotation property associated to an atomic entity, is directly transformed into one single triple while those complex logical constructors (the first six rows in Table 1), unlike the first strategy, are approximated. This strategy avoids the use of blank nodes in the RDF graph which may act as noise towards the correlation between entities when the embeddings are learned; but, the exact logical relationships are not kept in the resulting RDF graph. Moreover, the projection of membership and subsumption axioms (the seventh and eighth rows in Table 7.1) has two settings. In the first setting, the two involved atomic entities are transformed into one triple with the predicate of rdf:$type$ or $rdfs$:$subClassOf$. In the second setting, in addition to the above triple, one more triple which uses the inverse of rdf:$type$ or $rdfs$:$subClassOf$ is added. This enables a bidirectional walk between two entities with a subsumption or membership relationship on the transformed RDF graph, and would impact the corpus and the embeddings.

Both strategies can incorporate an OWL reasoner to compute the TBox classification and ABox realization before O is transformed into an RDF graph G. Such reasoning grounds the axioms of logical constructors and leads to explicit representation of some hidden knowledge [11].

Structure Document

The creation of the structure document aims at capturing both the graph structure and the logical constructors of the ontology. One option is computing random walks for each target entity in E with the RDF graph G. Each walk, which is a sequence of entity IRIs, acts as a sentence of the structure document. In order to implement the random walk algorithm, we first transform the RDF graph G into a directed single relation graph G'. More specifically, for each RDF triple $\langle X, r, Y \rangle$ in G, the subject X, the object Y and the relation r are transformed into three vertices, two edges are added from the vertex of X to the vertex of r and from the vertex of r to the vertex of Y respectively. Given one starting vertex, we fairly and randomly select the next vertex

from all its connected vertices, and iterate this "step" operation for a specific number of times to perform "walking".

OWL2Vec* also allows the usage of the Weisfeiler Lehman (WL) kernel [52] which encodes the structure of a sub-graph into a unique identity and thus enables the representation and incorporation of the sub-graph in a walk. For one vertex in the transformed single relation graph G', there is an associated sub-graph (neighbourhood) starting from this vertex. This sub-graph's WL kernel (identity) is called as this vertex's WL kernel. In the implementation, the original random walks are first extracted. For each random walk, the IRIs of the starting vertex and the vertices that are obtained from the relations are kept, but the IRIs of the none-starting vertices that are obtained from the subjects or objects with their WL kernels are replaced.

To capture the logical constructors, OWL2Vec* extracts all the axioms of the ontology and complements the sentences of the structure document. In the implementation, each ontology axiom is transformed into a sequence following the OWL Manchester Syntax , where the original built-in terms such as "subClassOf" and "some" are kept [11].

Lexical Document

The lexical document includes two kinds of word sentences. The first kind is generated from the entity IRI sentences in the structure document, while the second is extracted from the relevant lexical annotation axioms in the ontology. For the first kind, given an entity IRI sentence, each of its entities is replaced by its English label defined by $rdfs:label$. Note that the label is parsed and transformed into lowercase tokens, and those tokens with no letter characters are filtered out, before it replaces the entity IRI.

The second kind of word sentences are extracted from the textual annotations. They include two kinds: annotations by bespoke annotation properties such as $obo{:}IAO\_0000115$ ($definition$), $obo{:}IAO\_0010000$ ($hasaxiomlabel$) and $oboInOwl{:}hasSynonym$, and annotations by built-in annotation properties such as $rdfs{:}comment$ and $rdfs{:}seeAlso$. In the current OWL2Vec* implementation, we consider all the annotation properties of an ontology except for $rdfs{:}label$. The annotations by $rdfs{:}label$ are ignored in generating word sentences of the second kind because they are already considered in the word sentences of the first kind [11].

Combined Document

OWL2Vec*, also, extracts a combined document from the structure document and the entity annotations, so as to preserve the correlation between entities (IRIs) and words in the lexical information. On the one hand this would benefit the embeddings of the IRIs with the semantics of words. On the

other hand, the association with IRIs would incorporate some semantics of the graph structure into the words' embeddings. This may also add noise to the correlation between words and negatively impact the words' embeddings.

Two strategies are dealing with each IRI sentence in the structure document. The first strategy is to randomly select an entity in an IRI sentence, keep the IRI of this entity, and replace the other entities of this sentence by their lowercase word tokens extracted from their labels or IRI names as in the creation of the lexical document. The other strategy is traversing all the entities in a IRI sentence. For each entity, it generates a combined sentence by keeping the IRI of this entity, and replacing the others by their lowercase word tokens as in the random strategy. Thus for one IRI sentence, it generates m combined sentences where m is the number of entities of the IRI sentence [11].

Embeddings

After the generation of these three documents, OWL2Vec* merges the structure document, the lexical document and the combined document as one document, and then uses this document to train a Word2vec model with the skip-gram architecture. The hyper-parameter of the minimum count of words is set to 1 such that each word or entity (IRI) is encoded as long as it appears in the documents at least once. We can also pre-train the Word2vec model by a large and general corpus but this may be noisy and play a negative role in a domain specific task. OWL2Vec* is compatible with other word embedding or sequence feature learning methods, too.

With the trained word embedding model, OWL2Vec* calculates the embedding of each IRI (V_{iri}) and each word (V_{word}). The vectors can be used independently or can be concatenated and represent the embeddings of each entity. The embedding size is set before the training.

Chapter 8

Experimental study

In this chapter, we compare the performance of pretrained bert-based models with their semantic enrichment with the use of knowledge representation tools for two natural language processing tasks: information retrieval and text classification on medical papers. Firstly, we investigate the enrichment of these models with owl embeddings produced by the framework OWL2Vec* [11], which was described in Section 7.2.1, on the information retrieval task. Unfortunately, we prove that OWL2Vec* is unable to represent correctly the terms of SNOMED CT ontology, a terminology of medical terms, because it consists of a large number of concepts. We prove this, also, on the classification task. As a result, we decide to examine the enrichment of bert-based models with a concept filter, depending on terms in SNOMED CT, which in fact achieves better performance than simple bert-based models.

In the following sections we first describe the experimental settings for the evaluation of the methods in both tasks (Section 8.1), and then we present the results obtained (Section 8.2).

8.1 Experimental Settings

In this section, we provide the details about our experimental settings to make all the experiments reproducible.

8.1.1 Data Description

While it is easy to measure the performance of algorithms for classification problems, it is often hard to measure the performance of information retrieval systems. The main difficulty is finding an appropriate test dataset with a large number of documents and queries.

max #documents in OHSUMED	1	2	3	4	5	6	>6	Total
queries	0	1	1	0	0	0	61	63

Table 8.1: Query - document set size for Information Retrieval in OHSUMED dataset

For evaluation in both tasks we use subsets of the OHSUMED dataset to assess the improvement our method can achieve for document retrieval and classification compared to two different BERT models, used as baselines. The OHSUMED test collection is a subset of the MEDLINE database, the online medical information database, consisting of titles and/or abstracts of 270 medical journals over a five-year period (1987-1991). The available fields are the title, abstract, MeSH indexing terms, author, source and publication type but we use only the title and the abstract for both tasks. In the following sections we analyze the subsets of OHSUMED that were used in each task.

8.1.1.1 Information Retrieval

To evaluate the contribution of OWL2Vec* to Biomedical Information Retrieval System, we use a subset of the OHSUMED test collection as it was used for the TREC-9 Filtering Track. The test collection consists of 327,113 documents and it was built as part of a study assessing the use of MEDLINE by physicians in a clinical setting. Novice physicians using MEDLINE generated 106 queries. Only a subset of these queries was used in the TREC-9 Filtering Track. For the evaluation step we use a subset of 63 of the original query set developed by Hersh et al. for their IR experiments, each query was replicated by four searchers, two physicians experienced in searching and two medical librarians. The results were assessed for relevance by a different group of physicians, using a three-point scale: definitely, possibly or not relevant. In Table 8.1 we present a more detailed breakdown of query-document sets. As we discuss in the results, two queries are related with a low number of documents and this can have a negative impact on performance.

8.1.1.2 Classification

For the classification task we use a smaller OHSUMED test collection which consists of 30,590 documents. We refer to this subset of OHSUMED dataset as OHSUMED-CL. Each document belongs to one or more of the 23 categories. In Table 8.2 we present how many documents belong to each class. We can see that the dataset is unbalanced, which affects negatively our system. We mainly focus on two classes: Musculoskeletal Diseases (C05) and

Code	Name	#documents
C01	Bacterial Infections and Mycoses	2343
C02	Virus Diseases	1032
C03	Parasitic Diseases	393
C04	Neoplasms	5607
C05	Musculoskeletal Diseases	1500
C06	Digestive System Diseases	2691
C07	Stomatognathic Disease	473
C08	Respiratory Tract Diseases	2329
C09	Otorhinolaryngologic Diseases	659
C10	Nervous System Diseases	3504
C11	Eye Diseases	924
C12	Urologic and Male Genital Diseases	2316
C13	Female Genital Diseases and Pregnancy Complications	1462
C14	Cardiovascular Diseases	5323
C15	Hemic and Lymphatic Diseases	1141
C16	Neonatal Diseases and Abnormalities	977
C17	Skin and Connective Tissue Diseases	1446
C18	Nutritional and Metabolic Diseases	1678
C19	Endocrine Diseases	714
C20	Immunologic Diseases	2754
C21	Disorders of Environmental Origin	2706
C22	Animal Diseases	454
C23	Pathological Conditions, Signs and Symptoms	8597

Table 8.2: Number of documents at each class (code of class, name of class, number of documents) for Classification

Endocrine Diseases (C19). The first class consists of 1500 documents while the second 714.

8.1.1.3 SNOMED CT

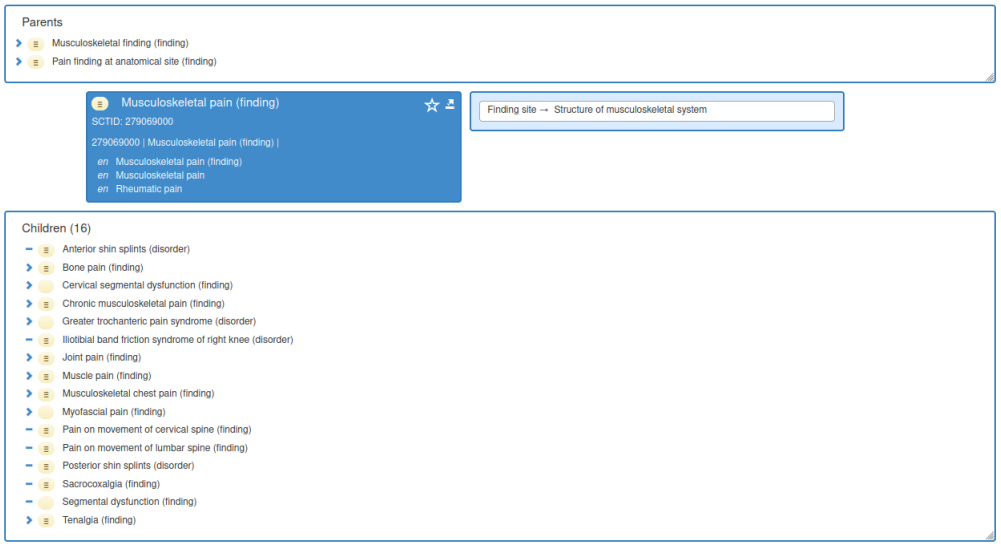


Figure 8.1: Example of the concept "Musculoskeletal pain (finding)" in SNOMED CT. The concept has 2 parents (Is-A relationship), 16 children, 1 attribute relationship (Finding site) and 3 synonyms. Source: SNOMED CT Browser (Fig. in [1]) .

SNOMED CT[1] (International Edition) [18] is an international terminology, a collection of medical terms, and their synonyms, descriptions, etc., with an underlying description logic formal model. It consists of more than 350,000 concepts and covering clinical findings, symptoms, diagnoses, procedures, body structures, organisms and other etiologies, substances, pharmaceuticals, devices and specimens among others. Developed by SNOMED International, a not-for-profit organisation based in the UK, it contains clinical knowledge that can complement textual information, and help us process new documents. Its core components include concepts, descriptions, and relationships.

Healthcare professionals in recording information can use different clinical terms that mean the same clinical 'thought'. SNOMED CT supports this by allowing more than one clinical term (description) for the same clinical 'thought' (concept). The concept is the basic building block in SNOMED CT and each concept has a unique ID (Code). In SNOMED CT there are, also, two commonly used description types, Fully Specified Name (FSN)

[1] `https://www.snomed.org/snomed-ct/five-step-briefing`

and Synonym (S). The FSN is the unique, unambiguous description of a concept while synonyms allow for different concepts to be used that have the same clinical meaning. Concepts are also associated with other concepts using relationships. These relationships are used to define and model in a logical manner the concepts. There are two types of relationships that exist in SNOMED CT, the 'Is-A relationship' and the attribute relationship. The 'Is-A relationship' which relates a concept to more general concept(s) is often known as the parent-child relationship. Each active child concept has at least one parent concept in its hierarchy but can have more than one. Concepts can also be further defined using an attribute relationship. Attribute relationships are an association between two concepts that specifies a defining characteristic of one of the concepts (the source of the relationship). Each attribute relationship has a name (the type of relationship) and a value (the destination of the relationship), all of which are concepts in their own right.

Figure 8.2: SNOMED CT Hierarchy. Source: SNOMED CT Browser (Fig. in [1]) .

SNOMED CT concepts are organised into 19 distinct hierarchies, each of which cover different aspects of healthcare. Concepts are organized from the general to the more detailed. This allows detailed clinical data to be recorded

and later accessed or aggregated at a more general level [43].

8.1.2 Data Preprocessing

We understand that preprocessing is necessary for both tasks. One of the challenges in terms of transferring the SNOMED-CT concepts to raw text, is to be able to identify the relevant terms in text. For this purpose, we employ the MetaMap tool [4], which maps biomedical text to the UMLS metathesaurus. Upon identifying the text spans that correspond to UMLS concepts, we use a mapping between UMLS and SNOMED concepts in order to incorporate the SNOMED knowledge. As a result, for each document apart from the abstract and the title we have the annotated terms from SNOMED CT.

In the information retrieval task, we, also, make experiments by splitting the abstract of each document of the OHSUMED dataset. More particularly, we split the abstract every two sentences and each subtext eventually contains the title and two sentences of the abstract. For example, the document with id "91005637" consists of the title "Gastrointestinal tuberculosis. Report of four cases." and the abstract "Gastrointestinal tuberculosis is a rare disease in the United States. Correct identification is often delayed because it is not considered early on in the differential diagnosis. Four patients with gastrointestinal tuberculosis and the symptoms, diagnosis, complications, and treatment of the disease are discussed. Gastrointestinal tuberculosis should be considered in Asian immigrant patients who present with symptoms and signs of inflammatory bowel disease.". After splitting, the document consists of two subtexts. The title belongs to both subtexts. As a result, the first subtext consists of the title, the first and the second sentence "Gastrointestinal tuberculosis. Report of four cases. Gastrointestinal tuberculosis is a rare disease in the United States. Correct identification is often delayed because it is not considered early on in the differential diagnosis." and the second subtext consists of the title, the third and the fourth sentence "Gastrointestinal tuberculosis. Report of four cases. Four patients with gastrointestinal tuberculosis and the symptoms, diagnosis, complications, and treatment of the disease are discussed. Gastrointestinal tuberculosis should be considered in Asian immigrant patients who present with symptoms and signs of inflammatory bowel disease.". This split improves the performance of our system as we will discuss later. We henceforth refer to this form of the dataset as OHSUMED-S dataset while the OHSUMED dataset, which contains the abstracts without split, is referred as OHSUMED dataset. No splitting to the queries dataset is applied.

Furthermore, for the information retrieval task in some experiments we

decide to enhance the SNOMED CT ontology with the documents themselves. Firstly, we create a new ontology, which contains the SNOMED CT terminology, new concepts with the id of each document, two data properties of each document, which contain its abstract and its title, and one object property, which actually connects each document with its annotated terms of SNOMED CT. We refer to this ontology as SNOMED CT-D. We also create a similar ontology with OHSUMED-S dataset. We refer to this ontology as SNOMED CT-DS and it contains the SNOMED CT terminology, new concepts with the id of each document and the id of each subtext of each document, one data property with the title of each document, one object property, which connects each document with its subtexts, one data property with the text of each subtext and one object property which connects each subtext of the document with its annotated terms of SNOMED CT.

8.1.3 Platform

We run our experiments on the SPOCK server of NTUA which has an Intel Xeon E5-2620 v4 CPU running at 2.10 GHz with 62 GB of RAM.

8.1.4 Implementation

We implement both tasks in Python. To load and modify SNOMED CT ontology, we use OWLready2 package. For searching and navigating the SNOMED CT hierarchy we use PyMedTermino2 module. We build and train all the following models in TensorFlow.

8.1.4.1 Transformer's Models Embeddings

We opt for two BERT embeddings trained on different domains, with demonstrated high performance in downstream classification tasks. Specifically we choose the original BERT model (Section 6.2), trained on Wikipedia and BookCorpus, hence fine-tuned for the generic domain and BioBERT (Section 6.3.4) trained on Pubmed papers, hence fine-tuned on the biomedical domain. We, also, use Longformer (Section 6.3.3), a transformer model pre-trained on long documents. For generating the embeddings of these models, we use the base models from HuggingFace's transformers: bert-base-uncased, biobert-v1.1 and longformer-base-4096 model respectively. To form a single vector representation for each document, we produce a mean pooling operation to the output vector of the models. In this way, we produce the embeddings for each document using their title and their abstract, if it exists (OHSUMED

dataset). We, also, generate the embeddings for the documents based on the splitting (OHSUMED-S dataset).

8.1.4.2 OWL Embeddings

To produce the embeddings of SNOMED CT, we use OWL2Vec*, a state-of the art system that creates embeddings from both the entities and the lexical information that appears in an ontology. We have, already, described how OWL2Vec* works in Section 7.2.1.

The algorithm accepts an ontology as input and produces embeddings as output. As we have analyzed previously, we have three different ontologies and by extension three different inputs of OWL2Vec* (SNOMED CT, SNOMED CT-D, SNOMED CT-DS). The algorithm first generates random walks over the ontology to extract structural, lexical, and semantic information in order to create a corpus of IRI and word sequences. As our ontology is very large, the option of Weisfeiler Lehman sub-tree kernel makes OWL2Vec* very slow and demands a lot of memory. As a result, we decide to choose random walks with 3 as walking depth, which is the length of the walk.

For the corpus that is fed to the word embedding model we use two different document settings of OWL2Vec*. The first is the Structure Document, D_s. It is composed of IRI sequences captured from the walks as well as the axioms, or relationships, between the classes within the ontology. The second document configuration is the Lexical Document, $D_{s,l}$, which replaces the IRIs of the structural document with the entity labels.

The corpus is then fed to the word embedding model Word2vec to create IRI and word (token) vector representations, V_{iri} and V_{word}. For the embedding model, three dimensions are tested (80, 100, 200) if no pre-training is adopted, and otherwise set to be consistent with the pre-trained model. For training the Word2vec without pre-training the window size is set to 5, the minimum count of words is set to 1; the number of epochs is set to 10. The Word2vec pre-training (with a dimension of 100) uses the latest English Wikipedia article dump and for fine-tuning this model the number of epochs is set to 100.

8.1.4.3 Information Retrieval

The goal is to compare the performance of the three different transformer's models with their enhancement with OWL2Vec*. Depending on the test collection (OHSUMED and OHSUMED-S) and the ontology that is used as input in OWL2Vec* (SNOMED CT, SNOMED CT-D, SNOMED-CT-DS), the experiments in the information retrieval task are organized as follows.

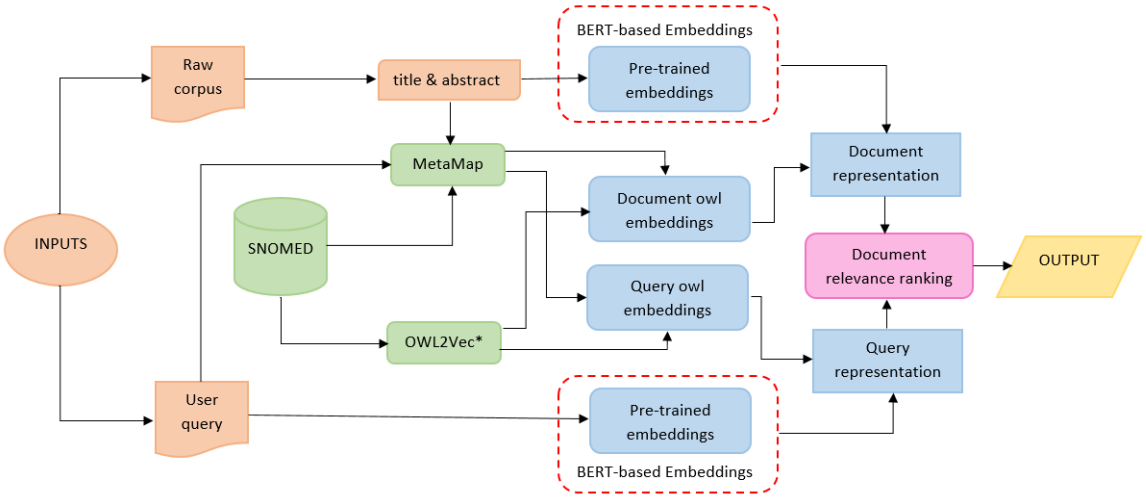


Figure 8.3: Summary of First System Architecture (IR).

(i) In the first group of experiments we first produce the transformer's models embeddings of each document of the OHSUMED dataset as described in Section 8.1.4.1. We then generate the owl embeddings of SNOMED CT from OWL2Vec* using different settings on each experiment (embedding size, document settings). In some experiments we use the IRI vector representation V_{iri}, in others the word vector representations V_{word} and in some others we concatenate these two vectors $V_{iri,word}$ for generating the owl embeddings of each document. More particularly, we create the owl embeddings for each document by taking the mean of the owl embeddings of the annotated terms of each document, which have been mapped to SNOMED CT after ignoring the terms that appear in all documents more than 30,000 times and all the stop words. The final embeddings of each document is a vector which contains both vector embeddings by concatenating the embeddings of a transformer-based model (BERT or BioBERT or Longformer) and the owl embeddings of this document. In a similar way we produce the embeddings of each query (both transformer-based embeddings and owl embeddings). We then rank the documents' relevance for each query by calculating the distance between two vector representations (embeddings) in order to retrieve documents relevant to the query. We use cosine similarity for the distance estimation, so if we assume that d_i is the document vector and q_i is the query vector then the relevance score is calculated as:

$$\text{relevance}(d_i, q_j) = \frac{d_i \cdot q_j}{\|d_i\| \cdot \|q_j\|} \tag{8.1}$$

In that way, for each query we have calculated its cosine similarity distance with every document.

(ii) In the second group of experiments the difference is that apart from generating the embeddings of each document of OHSUMED dataset, we also produce both transformer-based embeddings and owl embeddings after splitting the abstracts of the documents as described in Section 8.1.2 (OHSUMED-S). In this way, for every subtext of the document we produce its transformer-based embeddings and its owl embeddings from the annotated terms that appear in this subtext. For each subtext of the document, we concatenate the respective embeddings (transformer-based and owl). As a result, to each document correspond the embeddings of all the splits of the abstract of the document and the embeddings of the whole abstract of the document. We refer to this set of embeddings as S. If the number of sentences in the abstract of a document is n_s and the symbol // refers to the division which rounds down the answer and returns a whole number, the number of vectors (embeddings) corresponding to this document is $n_s//2 + 1$ if n_s is even and $n_s//2 + 2$ if n_s is odd. The embeddings of each query are produced in the same way as previously. For ranking a document's relevance for each query we calculate the cosine similarity between the query's vector representation and all the embeddings vectors of the document (embeddings from the splits and from the whole document). For each pair of document-query the maximum cosine similarity of these distances is the distance between the document and the query.

(iii) In the third group of experiments, we produce the transformer's models embeddings of each document of the OHSUMED dataset as described in Section 8.1.4.1 but the owl embeddings of each document are calculated differently. More specifically, as input in OWL2Vec* we use SNOMED-D and so we take the owl embeddings of each document by searching their V_{iri}. The final embeddings of each document is similarly a vector which contains both vector embeddings by concatenating the embeddings of a transformer-based model and the owl embeddings of this document. The queries embeddings are generated as previously and the ranking of the documents' relevance for each query is calculated as in the first experiments.

(iv) In the fourth group of experiments, we combine the approaches of the second and the third group of experiments. As input in OWL2Vec* we use SNOMED-DS and so we take the owl embeddings of each document and their splits by searching their V_{iri}. The transformer-based embeddings are produced as descibed in the second group of experiments. The final embeddings of each subtext is the concatenation of the respective embeddings. The rest of the process is the same as in the second group of experiments.

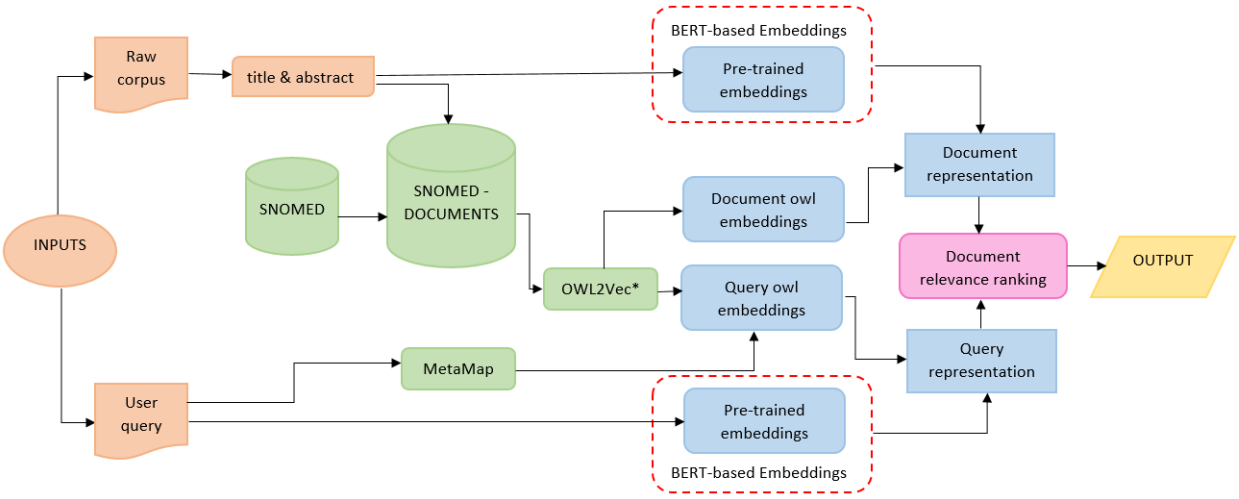


Figure 8.4: Summary of Second System Architecture (IR).

8.1.4.4 Classification

For the classification task, as we have already described, we use OHUSMED-CL dataset. The goal is to compare the performance of bert-based models with their enhancement with SNOMED CT. The dataset is split into 15% training set and 85% test set. All the following models consist of three layers: one input layer with relu as activation function, one hidden layer with 200 units with ReLU as activation function and one output layer with sigmoid as activation function. The layers are densely connected, or fully connected. We use as optimizer adam optimizer and as loss function binary cross-entropy loss. For training the models, the number of epochs is set to 2. In fact we investigate 3 different systems.

(i) Firstly, we test if OWL2Vec* enhances the bert-based models on the multilabel classification. We compare the performance of plain bert-

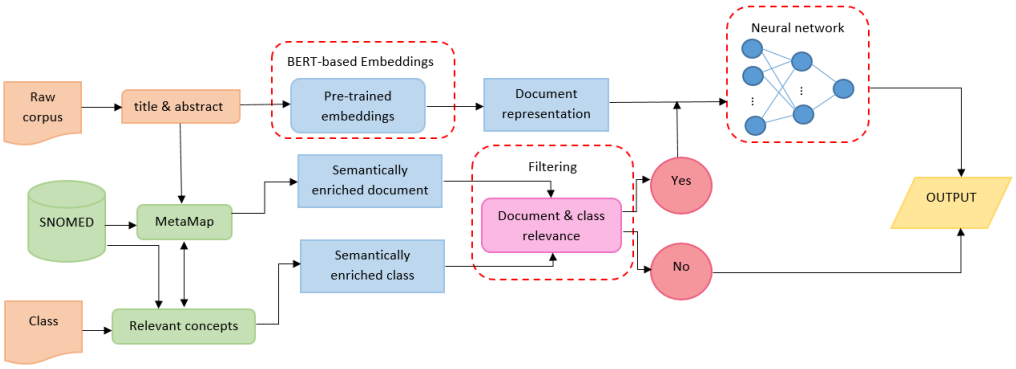


Figure 8.5: Summary of System Architecture (Classification).

based models with their enhancement with owl embeddings. We first build a neural network with 768 units in the input layer and 23 units in the output layer and only the bert-based embeddings of each document are fed into the network. Then, we enhance the bert-based embeddings with owl embeddings. The embeddings for each document are generated by concatenating its bert-based embeddings and owl embeddings. The owl embeddings are produced like the owl embeddings in the first group of experiments with an embedding size 100 of word vector representations and by using the $D_{s,l}$ document. The embeddings with a total size 868 (768+100) are fed into a neural network with 868 units in the input layer and 23 in the output layer. The output indicates to which classes belong each document.

(ii) Then, we test another system and we focus on two classes: Musculoskeletal Diseases (C05) and Endocrine Diseases (C19). On both classes we apply a concept filter on the documents. For each class we search the terms of SNOMED CT that include the words "musculoskeletal" and "endocrine" respectively. We choose to search for these words instead of searching "musculoskeletal disease" and "endocrine disease" as we observe that these choices limit a lot the terms of SNOMED CT and mainly focus on terms associated with the medical history of each disease. After finding the related concepts, we find their synonyms by using the MetaMap tool as described in Section 8.1.2. We navigate the SNOMED CT hierarchy to identify the parents, children and descendants of these concepts. We try different depths in SNOMED CT hierarchy. We refer to these two sets of concepts as CL_j where $j = 1, 2$. We apply this concept filter on documents in the following way: for

every document that all its term $c \in CD_i$ such that $c \notin CL_j$ we suppose that the document does not belong to the specific class. Otherwise, we suppose that the document belongs to the class. We, also, investigate the use of different thresholds on the value of the concept filter to filter the initial set of documents. In Table 8.3 we report an explanation of the different depths and thresholds of the concept filter that were investigated for the class C05. The depths and the thresholds for C19 are similar. As a result, we compare the performance of plain bert-based models with the performance of the concept filter.

(iii) Furthermore, we investigate the use of this concept filter with bert-based models. In fact, we compare the performance of plain bert-based models with their enhancement with the concept filter. We build a neural network with 768 units in the input layer and 1 unit in the output layer. Only the bert-based embeddings of each document are fed into the network. As the dataset is imbalanced, we also use a bias in the output layer, calculated as log (pos/neg) where pos is the number of documents that belong to the class and neg the number of documents that do not belong to the class. The output indicates if the document belongs to the class or not. For every document that is tested whether it belongs or not to the class, we first apply to it the concept filter. If all its term $c \in CD_i$ such that $c \notin CL_j$, the document does not belong to the class. Otherwise, the model predicts if the document belongs to the class. Because our dataset is imbalanced and the number of documents belonging to the class is small, we apply the concept filter before splitting the dataset into train and test set. The remaining set is split into train and test set and its set is fed into the model. For calculating the performance of the whole system, we take into account the predictions of the model for the test set and the set with the documents that have not passed the concept filter which are considered not to belong to the specific class.

8.1.5 Metrics

8.1.5.1 Information Retrieval

In the information retrieval task we base our evaluation on the normalised discounted cumulative gain (nDCG) metric, used to assess the model's ranking of relevant papers pertaining to a set of queries Q. It is defined for position $k \in \{0, 1, ..., N\}$:

Depths and Thresholds	Explanation
Depth 0	terms and synonyms of "musculoskeletal"
Depth 1	parents and children of Depth 0
Depth 1 & Ancestors	children and ancestors of Depth 0
Depth 2	parents and children of Depth 1
Depth 2 & Ancestors	children and ancestors of Depth 1
Threshold 1	at least 1 term of concept filter in the document
Threshold 2	at least 2 terms of concept filter in the document
Threshold 3	at least 3 terms of concept filter in the document

Table 8.3: Explanation of Depths and Thresholds for Concept Filter in class "Musculoskeletal Diseases"

$$\text{nDCG}_k = \frac{1}{Q}\sum_{q=1}^{Q}\frac{IDCG_k^{(q)}}{DCG_k^{(q)}}, \qquad \text{for} \quad \text{DCG}_k^{(q)} = rel_1^{(q)} + \sum_{i=2}^{k}\frac{rel_i^{(q)}}{\log_2(i)} \tag{8.2}$$

where $IDCG$ denotes the ideal and highest possible DCG and $rel_i^{(q)}$ refers to the relevance of the i^{th} result ranked according to query q.

8.1.5.2 Classification

In the classification task we base our evaluation on f1-score but we also calculate the precision and the recall metrics. For the binary classification (ii) and (iii) we calculate these metrics for each class separately while for the multilabel classification (i) the metrics are computed separately for each label and then these label-wise metrics are aggregated. The Precision, Recall and F1-Score Metrics are described in the following equations:

$$\text{Precision} = \frac{t_p}{t_p + f_p} \tag{8.3}$$

$$\text{Recall} = \frac{t_p}{t_p + f_n} \tag{8.4}$$

$$\text{F1-Score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{8.5}$$

where tp the number of outcomes where the system correctly predicts that the documents belong to the class, tn the number of outcomes where the system correctly predicts that the documents do not belong to the class, fp the number of outcomes where the system incorrectly predicts that the

Transformer Models	Plain	OWL2Vec*
BERT	0.114	0.073
BioBERT	0.137	0.11
Longformer	0.008	0.013

Table 8.4: nDCG scores ($k = 5$) of Transformer models before (Plain) and after their enhancement with OWL2Vec* ($D_{s,l}, embedding\_size : 100, V_{word}$).

documents belong to the class and fn the number of outcomes where the system incorrectly predicts that the documents do not belong to the class.

8.2 Experimental Results

In this section, we present our experimental results. We first compare the approaches mentioned in the information retrieval task (Section 8.2.1) and then in the classification task (Section 8.2.2).

8.2.1 Information Retrieval

We first compare the results between the transformer's-based models and their enhancement with the owl embeddings produced by OWL2Vec* as described at (i) in Section 8.1.4.3. In Table 8.4 we report the experimental results of BERT, BioBERT and Longformer models in comparison with their results after concatenating their embeddings with the owl embeddings of OWL2Vec*. For producing these owl embeddings we choose the document setting $D_{s,l}$, 100 for embedding size and the word vector representations (V_{word}). As we expected BioBERT is better than BERT and Longformer as BioBERT is trained on medical papers. However, OWL2Vec* when combined with BERT and BioBERT worsens the results. Longformer seems not to have good results on both cases but its results have a slight improvement when combined with OWL2Vec*. We then experiment with the embedding size of OWL2Vec*. In Table 8.5 we report the results of Biobert with OWL2Vec* for the embedding sizes 80, 100 and 200. We can see that the nDCG score, when the embedding size is 80, is the best one and the worst is when the embedding size is 200. We also experiment with the document settings and vector representations of owl embeddings. We set an embedding size of 80. In Table 8.6 and in Figure 8.6 we compare BERT and BioBERT models with their combination of OWL2Vec* for different settings. We can see that we achieve the best scores when we use the iri vector representations V_{iri} of owl

Embedding Size	BioBERT & OWL2Vec*
80	0.126
100	0.11
200	0.091

Table 8.5: nDCG scores ($k = 5$) of BioBERT after its enhancement with OWL2Vec* for different embedding sizes ($D_{s,l}, V_{word}$).

Settings	BERT & OWL2Vec*	BioBERT & OWL2Vec*
$D_s + V_{iri}$	0.113	0.146
$D_{s,l} + V_{iri}$	0.114	0.145
$D_{s,l} + V_{word}$	0.112	0.126
$D_{s,l} + V_{iri,word}$	0.092	0.094

Table 8.6: nDCG scores ($k = 5$) of BERT and BioBERT after their enhancement with OWL2Vec* using different document and embedding settings ($embedding\_size : 80$).

embeddings and in fact OWL2Vec* improves slightly but not satisfactorily the bert-based models. The use of the concatenation $V_{iri,word}$ worsens both scores.

Furthermore, we compare the results between the approaches described at (i) and at (ii) in Section 8.1.4.3. In Table 8.7 we present the nDCG scores of BERT and BioBERT with or without their enhancement with OWL2Vec* using these two methods and two different embedding sizes. In OWL2Vec* we have used $D_{s,l}$ and V_{word}. We can see that splitting the documents in fact improves all models, especially BioBERT. It is, also, worth noting that the BERT model with splitting almost reaches the scores of BioBERT without splitting. In all cases, OWL2Vec* still does not improve the scores of bert-based models.

In Table 8.8 we compare the results between the approaches described at (iii) and at (iv) in Section 8.1.4.3. We represent the nDCG scores of BERT and BioBERT with or without their enhancement with OWL2Vec* using these two methods. In OWL2Vec* we have used $D_{s,l}$ and V_{iri} for the documents' vector representations and V_{word} for the queries' vector representations. We can conclude that by inserting the documents in SNOMED CT and then producing their owl embeddings, the nDCG scores for both BERT and BioBERT are slightly improved. Their splitting improves even more the models. We, also, report some other experiments with the approach (iv). In Table 8.9 we report

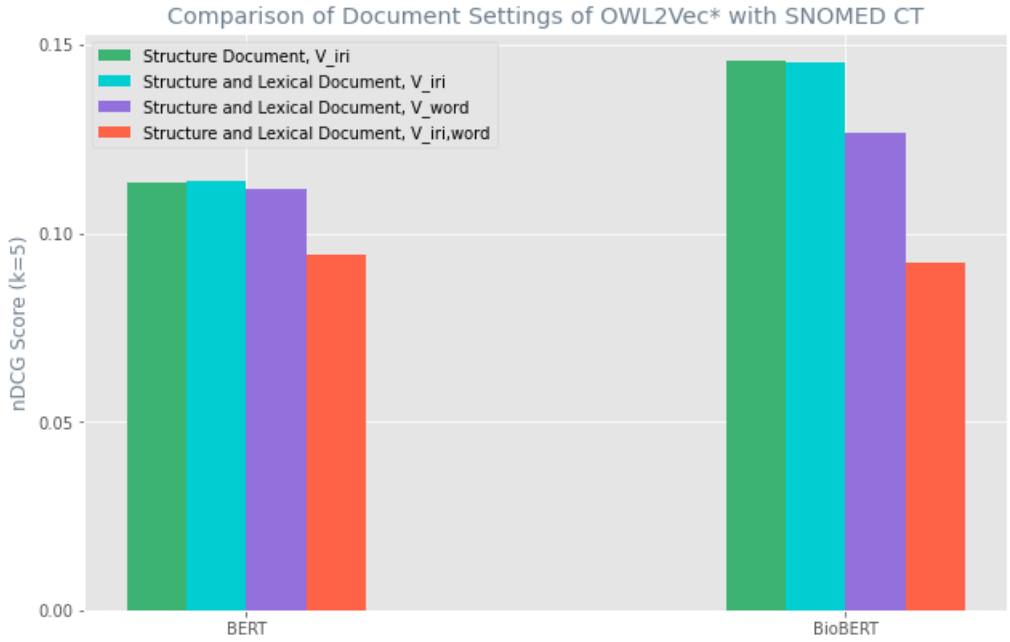


Figure 8.6: Results of the models BERT and BioBERT with OWL2Vec* using OHSUMED and SNOMED CT for different document and embedding settings ($embedding\_size$: 80)

the experiments with BioBERT and OWL2Vec* for two different embedding sizes (80, 100) while in Table 8.10 the experiments with BERT and BioBERT with pre-trained OWL2Vec*. We can see that the best embedding size is 100 while the scores of the pre-trained OWL2Vec* are worse. As a result we show that pre-trained can be noisy as explained in Section 7.2.1. In Table 8.11 and in Figure 8.7 we compare different settings of OWL2Vec* in the (iv) method. The vector representations V_{word} and V_{iri} refer to the representations of queries and V_{word} seem to represent them more appropriately. In Table 8.12 we compare the nDCG scores of BioBERT and BioBERT with OWL2Vec* (V_{word} for queries, 100 embedding size, $D_{s,l}$) for different values of k. In Figure 8.8 we present the best results of all approaches.

From the experimental results we can conclude that the best documents and embeddings settings of OWL2Vec* for the information retrieval of medical papers with the use of SNOMED CT are the following: the use of $D_{s,l}$, embedding size of 80 for SNOMED CT and 100 for SNOMED CT with the documents, iri vector representations V_{iri} for both queries and documents when we use SNOMED CT and word vector representations V_{word} for queries when we use SNOMED CT-D and SNOMED-DS. Splitting the abstracts into subtexts improves all the results, a method that was also proved that it

Model	Without Split (i)	With Split (ii)
BERT Plain	0.114	0.13
BERT & OWL2Vec* (80)	0.112	0.131
BERT & OWL2Vec* (100)	0.073	0.129
BioBERT Plain	0.137	0.194
BioBERT & OWL2Vec* (80)	0.126	0.182
BioBERT & OWL2Vec* (100)	0.11	0.168

Table 8.7: nDCG scores ($k = 5$) of BERT and BioBERT before and after their enhancement with OWL2Vec* using the two different datasets OHSUMED and OHSUMED-S ($D_{s,l}, embedding\_sizes : 80, 100, V_{word}$).

Model	Without Split (iii)	With Split (iv)
BERT Plain	0.114	0.13
BERT & OWL2Vec* (100)	0.124	0.141
BioBERT Plain	0.137	0.194
BioBERT & OWL2Vec* (100)	0.142	0.206

Table 8.8: nDCG scores ($k = 5$) of BERT and BioBERT before and after their enhancement with OWL2Vec* using the two different ontologies SNOMED CT-D and SNOMED CT-DS ($D_{s,l}, embedding\_size : 100$).

BioBERT & OWL2Vec*	nDCG Score
Embedding Size 80	0.198
Embedding Size 100	0.206

Table 8.9: nDCG scores ($k = 5$) of BioBERT after its enhancement with OWL2Vec* using the ontology SNOMED CT-DS for two different embedding sizes ($D_{s,l}$).

Model	nDCG Score
BERT & pre-trained OWL2Vec*	0.126
BioBERT & pre-trained OWL2Vec*	0.147

Table 8.10: nDCG scores ($k = 5$) of BERT and BioBERT after their enhancement with OWL2Vec* using the ontology SNOMED CT-DS ($D_{s,l}$).

Settings	BERT & OWL2Vec*	BioBERT & OWL2Vec*
$D_s + V_{iri}$	0.115	0.119
$D_{s,l} + V_{iri}$	0.116	0.132
$D_{s,l} + V_{word}$	0.141	0.206

Table 8.11: nDCG scores ($k = 5$) of BERT and BioBERT after their enhancement with OWL2Vec* using the ontology SNOMED CT-DS using different document and embedding settings ($embedding\_size$: 100).

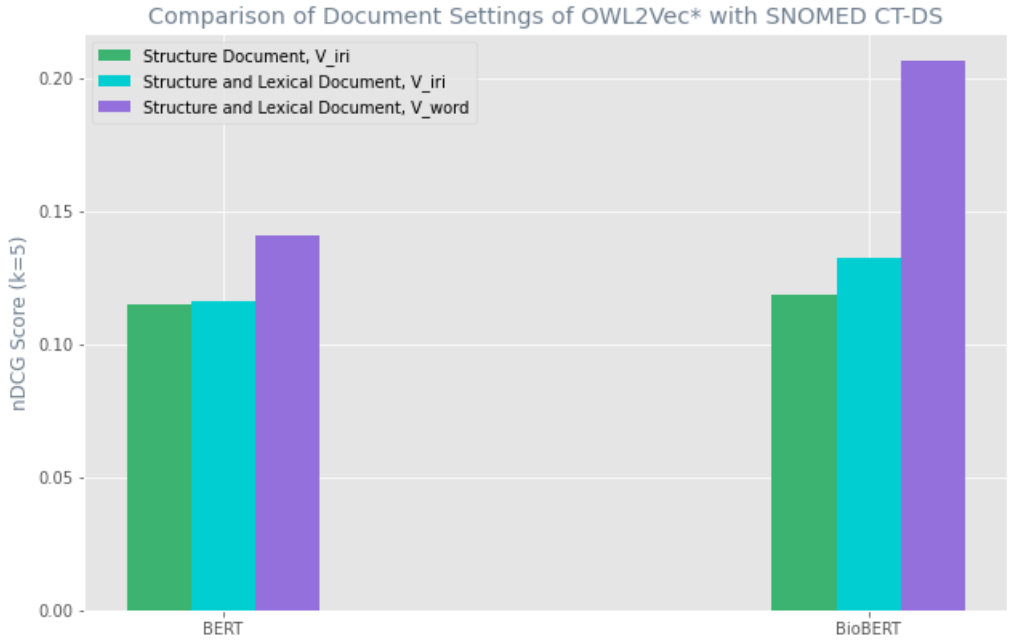


Figure 8.7: Results of the models BERT and BioBERT with OWL2Vec* using SNOMED CT-DS for different document and embedding settings ($embedding\_size$: 100)

improves the performance of models in text classification of other domains [56]. More specifically, Chi Sun et al. proposed that there are parts of a text that involves more insignificant information, for example the middle of the text. By using SNOMED CT-DS we have achieved slightly better performance with OWL2Vec* in comparison with plain bert-based models. In general, it seems that OWL2Vec* does not enhance the performance of bert-based models in this task. In fact, OWL2Vec* has been tested with medium (e.g. an events ontology) and small (e.g. pizza ontology) size ontologies and has shown good performance. However, it seems unable to perform well in larger

k	BioBERT	BioBERT & OWL2Vec*
5	0.194	0.206
10	0.164	0.173
20	0.149	0.154
50	0.154	0.158
100	0.174	0.181
1000	0.279	0.283

Table 8.12: nDCG scores for different values of k of BioBERT before and after its enhancement with OWL2Vec* using the ontology SNOMED CT-DS ($D_{s,l}, embedding\_size : 100$).

vocabularies, like SNOMED CT which has more than 350,000 concepts, and render correctly their lexical information and logical constructors. Ritchie et al. [48] expects that OWL2Vec* performance in OWL2Vec* will improve with larger ontologies in the future. It is important to mention that the low number of related documents per query significantly impacts the nDCG scores. We expect that we would see considerably higher scores for datasets with a larger number of related documents per query.

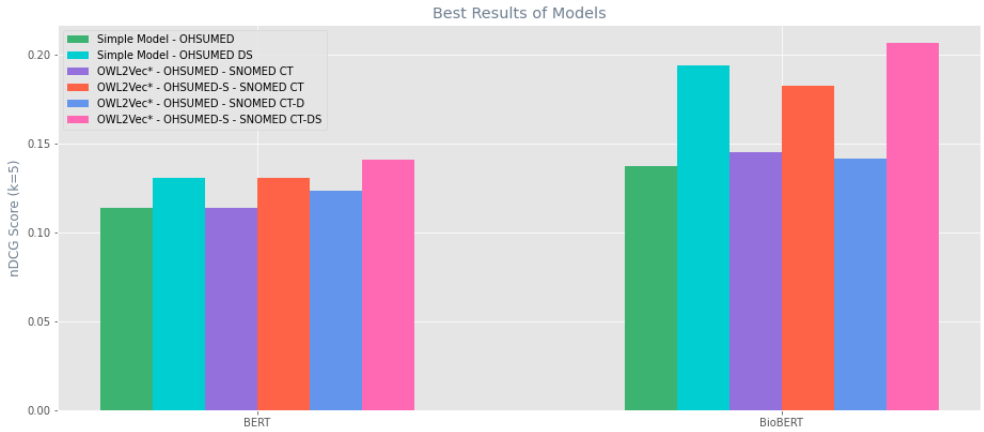
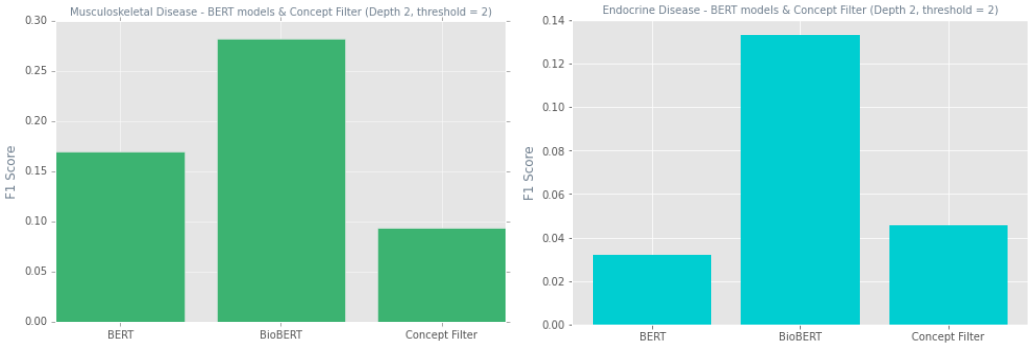


Figure 8.8: Best results of the models in information retrieval for different versions of dataset and ontology.

8.2.2 Classification

We decide to use OWL2Vec* in a different task, multilabel text classification as described in Section 8.1.4.4 (i), in order to prove that OWL2Vec*



(a) Class: Musculoskeletal Diseases (C05) (b) Class: Endocrine Diseases (C19)

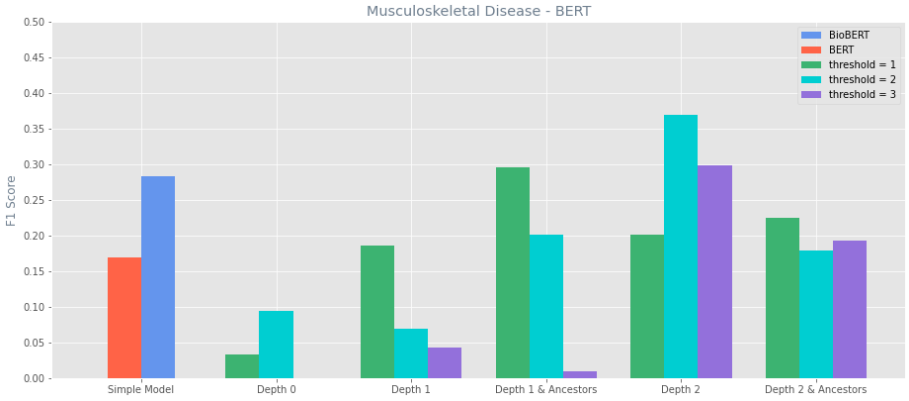
Figure 8.9: Comparison of f1-scores of BERT, BioBERT and concept filter for two classes.

cannot be applied well in SNOMED CT. The performance of bert-based models is not indeed improved with their enhancement with OWL2Vec*. As a result, we try a different approach in the binary classification task for two classes with the use of a concept filter described at (ii) and (iii).

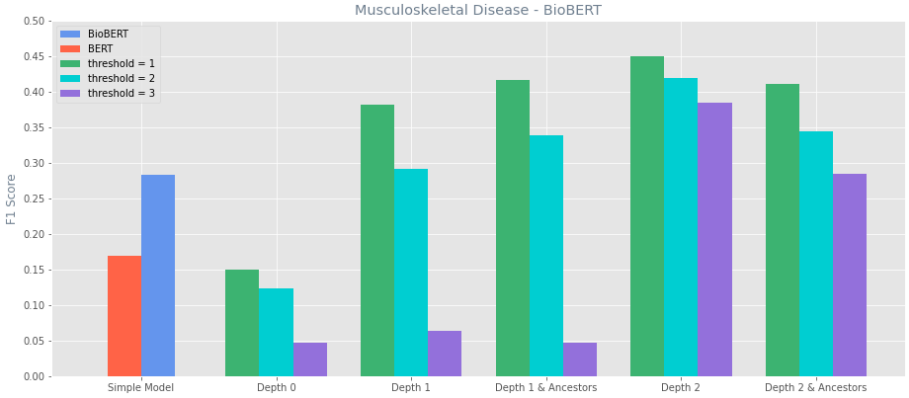
Firstly, we focus on the system (ii) which does not require a machine learning model. We just apply a concept filter on the documents without training a model. We use different depths while searching the parents and the children of a term in SNOMED CT and different thresholds in the concept filter as explained in Table 8.3. In Figure 8.9 we report the f1-scores of BERT, BioBERT and the concept filter for both classes for depth 2 and threshold 2. We can see that just by applying a concept filter in the dataset, our system achieves f1-scores that are close to f1-scores of BERT model. In fact, the f1-scores of the concept filter on the C19 class surpass those of BERT model.

We then continue with system (iii) which requires the use of a bert-based model, its training for 2 epochs and the use of the concept filter. We compare the bert-based models BERT and BioBERT with their enhancement with the concept filter for different thresholds and depths.

Regarding the class "Musculoskeletal Diseases" in Figure 8.10 and Figure 8.11 we can see that BERT with the use of some depths and thresholds of concept filter even surpasses simple BioBERT model. More specifically, the concept filter with Depth 1 & Ancestors - Threshold 1 and Depth 2 - Thresholds 2 and 3 in combination with BERT achieves better performance than BioBERT. We can also see that the scores of concept filter in combination with BioBERT model outperform the scores of simple BioBERT in many cases. Furthermore we notice that for small depths with the increase of threshold the f1-score of the system is reduced as there are less terms that are



(a) BERT Model

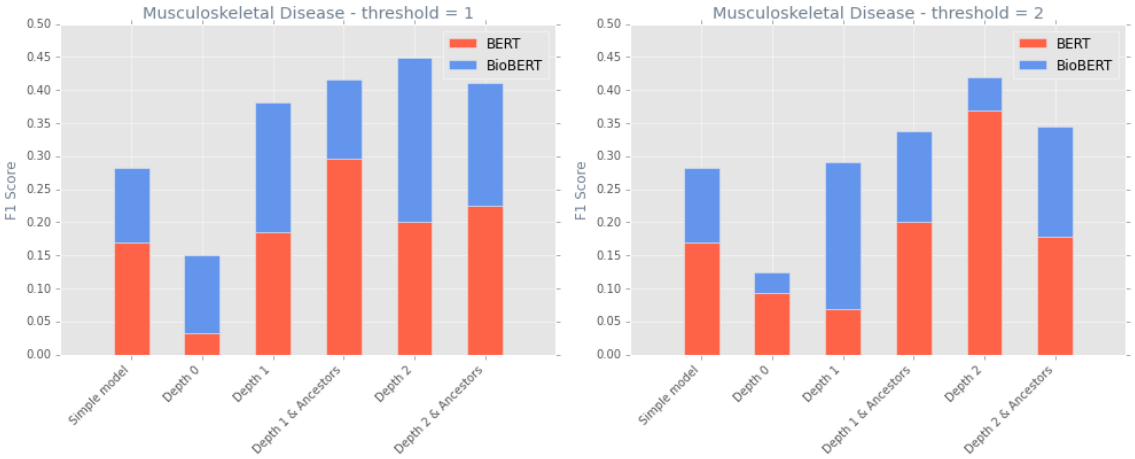


(b) BioBERT Model

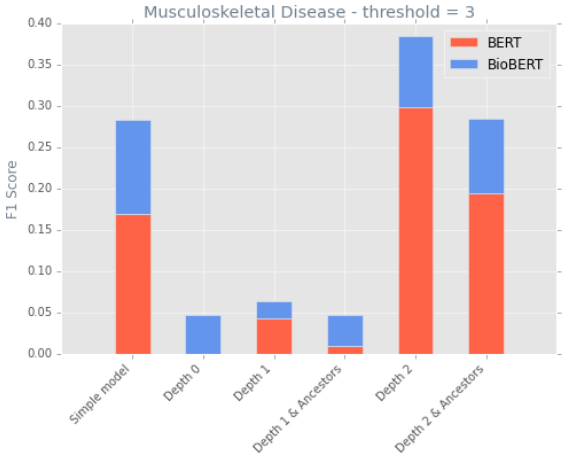
Figure 8.10: Comparison of f1-scores of simple BERT, BioBERT with their enhancement with the concept filter for "Musculoskeletal Diseases" Class.

found in SNOMED CT and as a result there are not many documents that pass the filtering and so many papers that belong to the class, are considered not to be. Respectively, filtering with Depth 2 & Ancestors seems to be worse than filtering with Depth 2 as many terms that belong to ancestors are included in many documents.

As regards the second class "Endocrine Diseases" we can see in Figure 8.12 and in Figure 8.13 that all of our systems outperform BERT model. Moreover, the concept filter with Depth 1 - Threshold 3 in combination with BERT achieves better performance than BioBERT. It is worth noting that concepts related with "endocrine" term are a lot more than those related with "musculoskeletal". That's why we achieve the best performance with a small depth and a big threshold. The concept filter with Depth 1 - Threshold 3 has also good scores with BioBERT but we also notice that even a concept filter with

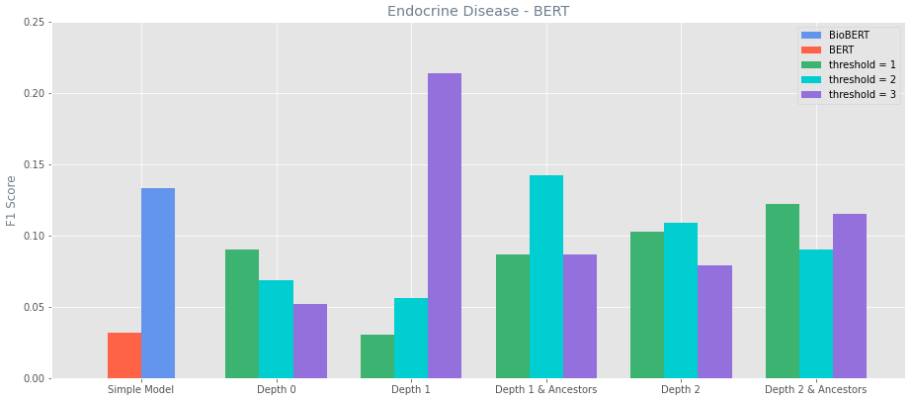


(a) Threshold 1 (b) Threshold 2

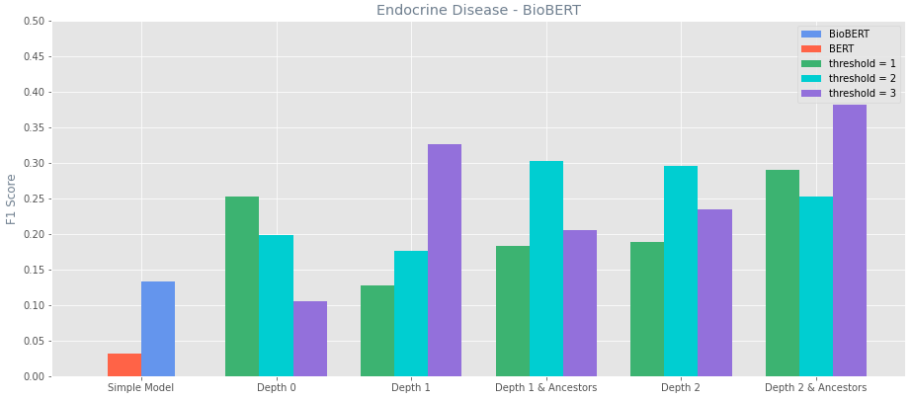


(c) Threshold 3

Figure 8.11: Comparison of f1-scores of simple BERT, BioBERT with their enhancement with the concept Filter for specific thresholds and various depths ("Musculoskeletal Diseases" Class).



(a) BERT Model

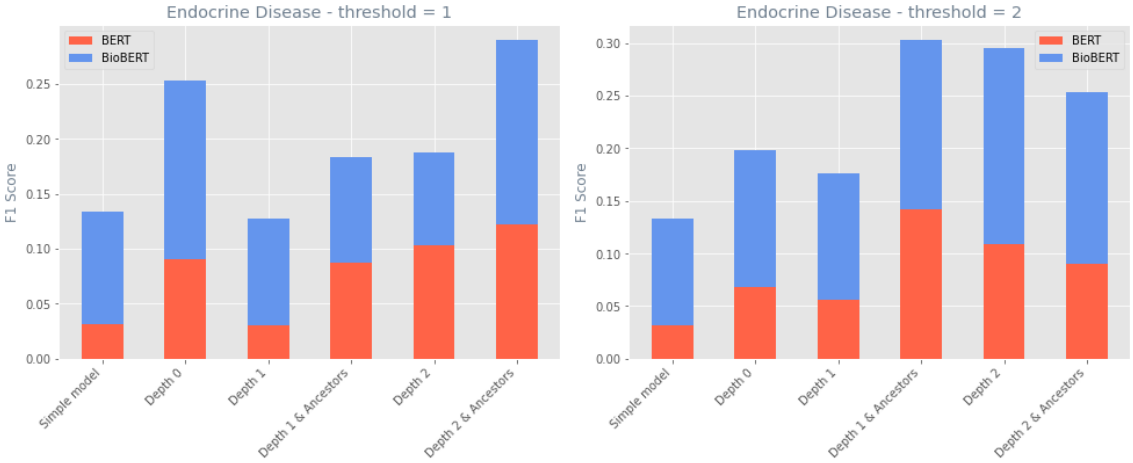


(b) BioBERT Model

Figure 8.12: Comparison of f1-scores of simple BERT, BioBERT with their enhancement with the concept filter for "Endocrine Diseases" Class.

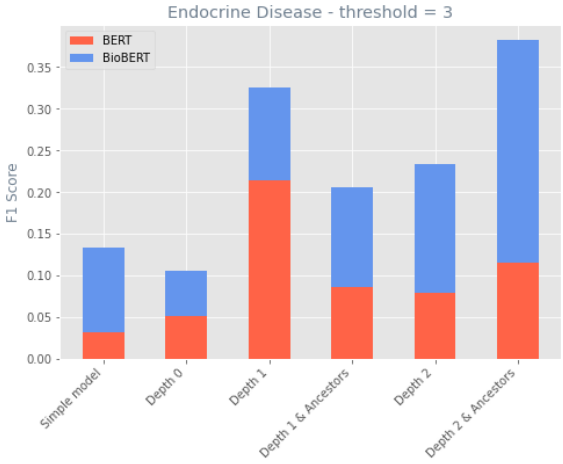
Depth 0 and Threshold 1 can outperform BioBERT.

We understand that the use of a concept filter with bert-based models can help them achieve better results at text classification task. In fact, we consider that with a more balanced dataset our system can accomplish better performance.



(a) Threshold 1

(b) Threshold 2



(c) Threshold 3

Figure 8.13: Comparison of f1-scores of simple BERT, BioBERT with their enhancement with the concept Filter for specific thresholds and various depths ("Endocrine Diseases" Class).

Chapter 9

Conclusion and Future Work

In this thesis, we explored options for improving unsupervised information retrieval on emerging queries and supervised classification in the biomedical domain. We investigated the use of SNOMED CT to improve the initial results of bert-based models. We first investigated the framework OWL2Vec* for both information retrieval and classification task but we found that currently the framework cannot be implemented efficiently for large ontologies, such as SNOMED CT. However, the performance is expected to improve with larger vocabularies in the future. In the classification task we also proposed a simple co-occurrence filtering method which accomplishes a very good performance in comparison with bert-based models. Indeed, we found that BERT-based results filtered using SNOMED CT surpass the performance of unfiltered BioBERT results.

The aforementioned outcomes provide solid basis for further experimentation into better exploitation of knowledge graphs and concept hierarchies as a means of boosting IR and Classification on new topics in an unsupervised manner. More importantly, these observations demonstrate that a specialized ontology, can successfully be applied to adapt out-of-domain models to a new domain. In the future, as for OWL2Vec* the framework would efficiently produce owl embeddings that express correctly the lexical information and logical constructors of a large ontology. Owl embeddings can significantly boost the performance of pretrained neural networks on natural language processing tasks. As for the concept filter, the exploration in more detail of the SNOMED CT hierarchy would enhance further the pretrained models. The position of a concept in the hierarchy, the size and the type of its neighbors would potentially allow us to identify further connections among classes and documents. The use of this filtering method can also be applied on other domains, such as the financial sector, with other specialized external knowledge sources.

Bibliography

[1] *Snomed ct browser*. https://browser.ihtsdotools.org/. Accessed: 2022-05-20.

[2] Admin, Jobs: *A simple introduction to sequence to sequence models*. "https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/", Accessed: 2022-02-15.

[3] Agosti, Maristella, Stefano Marchesin, and Gianmaria Silvello: *Learning unsupervised knowledge-enhanced representations to reduce the semantic gap in information retrieval*. ACM Trans. Inf. Syst., 38(4), sep 2020, ISSN 1046-8188. https://doi.org/10.1145/3417996.

[4] Aronson, Alan and François Michel Lang: *An overview of metamap: Historical perspective and recent advances*. Journal of the American Medical Informatics Association : JAMIA, 17:229–36, May 2010.

[5] Batut, Bérénice, Saskia Hiltemann, Andrea Bagnacani, Dannon Baker, Vivek Bhardwaj, Clemens Blank, Anthony Bretaudeau, Loraine Brillet-Guéguen, Martin Čech, John Chilton, Dave Clements, Olivia Doppelt-Azeroual, Anika Erxleben, Mallory Ann Freeberg, Simon Gladman, Youri Hoogstrate, Hans Rudolf Hotz, Torsten Houwaart, Pratik Jagtap, Delphine Larivière, Gildas Le Corguillé, Thomas Manke, Fabien Mareuil, Fidel Ramírez, Devon Ryan, Florian Christoph Sigloch, Nicola Soranzo, Joachim Wolff, Pavankumar Videm, Markus Wolfien, Aisanjiang Wubuli, Dilmurat Yusuf, James Taylor, Rolf Backofen, Anton Nekrutenko, and Björn Grüning: *Community-driven data analysis training for biology*. Cell Systems, 6(6):752–758.e1, jun 2018. https://doi.org/10.1016%2Fj.cels.2018.05.012.

[6] Beltagy, Iz, Matthew E. Peters, and Arman Cohan: *Longformer: The long-document transformer*, 2020.

[7] Bishop, Christopher M.: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006, ISBN 0387310738.

[8] Bordes, Antoine, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko: *Translating embeddings for modeling multi-relational data*. In Burges, C.J., L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger (editors): *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. https://proceedings.neurips.cc/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.

[9] Braga, Juliao, Joaquim Dias, and Francisco Regateiro: *A MACHINE LEARNING ONTOLOGY*. October 2020.

[10] Caroline Clabaugh, Dave Myszewski, Jimmy Pang: *Neural networks: The perceptron*. "https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Neuron/index.html", Accessed: 2022-02-15.

[11] Chen, Jiaoyan, Pan Hu, Ernesto Jimenez-Ruiz, Ole Magnus Holter, Denvar Antonyrajah, and Ian Horrocks: *Owl2vec*: Embedding of owl ontologies*, 2020. https://arxiv.org/abs/2009.14654.

[12] Cho, Kyunghyun, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio: *On the properties of neural machine translation: Encoder-decoder approaches*, 2014.

[13] Cybenko, G.: *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals, and Systems (MCSS), 2(4):303–314, December 1989, ISSN 0932-4194. http://dx.doi.org/10.1007/BF02551274.

[14] Daniel Jurafsky, James H. Martin: *Speech and Language Processing*. 2021.

[15] Dervakos, Edmund, Giorgos Filandrianos, Konstantinos Thomas, Alexios Mandalios, Chrysoula Zerva, and G. Stamou: *Semantic enrichment of pretrained embedding output for unsupervised ir*. CEUR Workshop Proc., 2846, 2021.

[16] Devlin, Jacob, Ming Wei Chang, Kenton Lee, and Kristina Toutanova: *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019.

[17] Donges, Niklas: *A guide to rnn: Understanding recurrent neural networks and lstm networks.* "https://builtin.com/data-science/recurrent-neural-networks-and-lstm", Accessed: 2022-02-15.

[18] Donnelly, Kevin: *Snomed-ct: The advanced terminology and coding system for ehealth.* Studies in health technology and informatics, 121:279–90, February 2006.

[19] Education, IBM Cloud: *Machine learning.* "https://www.ibm.com/cloud/learn/machine-learning", Accessed: 2022-03-05.

[20] Education, IBM Cloud: *Recurrent neural networks.* "https://www.ibm.com/cloud/learn/recurrent-neural-networks", Accessed: 2022-02-15.

[21] Ehrlinger, Lisa and Wolfram Wöß: *Towards a definition of knowledge graphs.* September 2016.

[22] Firth, J. R.: *A synopsis of linguistic theory 1930-55.* 1952-59:1–32, 1957.

[23] Gad", "Ahmed: *"a comprehensive guide to the backpropagation algorithm in neural networks"*, December 2021. "https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide", Accessed: 2022-02-18.

[24] Gonzalez, R.C. and R.E. Woods: *Digital Image Processing.* Pearson-/Prentice Hall, 2008, ISBN 9780131687288. https://books.google.gr/books?id=8uGOnjRGEzoC.

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

[26] Graves, Alex: *Generating sequences with recurrent neural networks*, 2014.

[27] Grover, Aditya and Jure Leskovec: *node2vec: Scalable feature learning for networks*, 2016. https://arxiv.org/abs/1607.00653.

[28] Guarino, Nicola, Daniel Oberle, and Steffen Staab: *What Is an Ontology?*, pages 1–17. May 2009.

[29] Harris, Zellig S.: *Distributional structure.* WORD, 10(2-3):146–162, 1954. https://doi.org/10.1080/00437956.1954.11659520.

[30] Hochreiter, Sepp and Jürgen Schmidhuber: *Long short-term memory.* Neural computation, 9:1735–80, December 1997.

[31] Holter, Ole Magnus, Erik Bryhn Myklebust, Jiaoyan Chen, and Ernesto Jiménez-Ruiz: *Embedding owl ontologies with owl2vec.* In *SEMWEB*, 2019.

[32] Janiesch, Christian, Patrick Zschech, and Kai Heinrich: *Machine learning and deep learning.* Electronic Markets, 31(3):685–695, apr 2021. https://doi.org/10.1007%2Fs12525-021-00475-2.

[33] Joos, Martin: *Description of language design.* Journal of the Acoustical Society of America, 22:701–707, 1950.

[34] Kalyan, Katikapalli Subramanyam, Ajit Rajasekharan, and Sivanesan Sangeetha: *Ammus : A survey of transformer-based pretrained models in natural language processing*, 2021.

[35] Kamali, Kaivan: *Deep learning (part 1) - feedforward neural networks (fnn) (galaxy training materials)*, June 2021. "https://training.galaxyproject.org/training-material/topics/statistics/tutorials/FNN/tutorial.html", [Online; accessed Tue Feb 22 2022].

[36] Karpathy, Andrej: *The unreasonable effectiveness of recurrent neural networks.* "http://karpathy.github.io/2015/05/21/rnn-effectiveness", Accessed: 2022-02-15.

[37] Lee, Jinhyuk, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang: *Biobert: a pre-trained biomedical language representation model for biomedical text mining.* Bioinformatics, Sep 2019, ISSN 1460-2059. http://dx.doi.org/10.1093/bioinformatics/btz682.

[38] Lillicrap, Timothy P and Adam Santoro: *Backpropagation through time and the brain.* Current Opinion in Neurobiology, 55:82–89, 2019, ISSN 0959-4388. https://www.sciencedirect.com/science/article/pii/S0959438818302009, Machine Learning, Big Data, and Neuroscience.

[39] Lin, Yankai, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu: *Learning entity and relation embeddings for knowledge graph completion.* In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, page 2181–2187. AAAI Press, 2015, ISBN 0262511290.

[40] Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov: *Roberta: A robustly optimized bert pretraining approach*, 2019.

[41] Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean: *Efficient estimation of word representations in vector space*, 2013. `https://arxiv.org/abs/1301.3781`.

[42] Newell, Allen: *A step toward the understanding of information processes: Perceptrons. an introduction to computational geometry. marvin minsky and seymour papert. m.i.t. press, cambridge, mass., 1969. vi + 258 pp., illus. cloth, $12; paper, $4.95.* Science, 165(3895):780–782, 1969. `https://www.science.org/doi/abs/10.1126/science.165.3895.780`.

[43] NHS Digital: *SNOMED CT: A user guide for General Practice.*

[44] Ostendorff, Malte, Peter Bourgonje, Maria Berger, Julian Moreno-Schneider, Georg Rehm, and Bela Gipp: *Enriching bert with knowledge graph embeddings for document classification*, 2019. `https://arxiv.org/abs/1909.08402`.

[45] Pennington, Jeffrey, Richard Socher, and Christopher Manning: *GloVe: Global vectors for word representation.* In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. `https://aclanthology.org/D14-1162`.

[46] Pilehvar, Mohammad Taher and Jose Camacho-Collados. 2020.

[47] Ristoski, Petar and Heiko Paulheim: *Rdf2vec: Rdf graph embeddings for data mining.* In *SEMWEB*, 2016.

[48] Ritchie, A., J. Chen, L. J. Castro, D. Rebholz-Schuhmann, and E. Jimenez-Ruiz: *Ontology clustering with owl2vec*.* In *Deep Learning meets Ontologies and Natural Language Processing (DeepOntoNLP2021)*, volume 2918, pages 54–61. CEUR Workshop Proceedings, July 2021. `https://openaccess.city.ac.uk/id/eprint/25933/`, Copyright © 2021 This paper is reproduced under the Creative Commons License Attribution 4.0 International (CC BY 4.0).

[49] Salton, G., A. Wong, and C. S. Yang: *A vector space model for automatic indexing.* Commun. ACM, 18(11):613–620, nov 1975, ISSN 0001-0782. `https://doi.org/10.1145/361219.361220`.

[50] Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf: *Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter*, 2020.

[51] Schuster, Mike and Kuldip Paliwal: *Bidirectional recurrent neural networks*. Signal Processing, IEEE Transactions on, 45:2673 – 2681, December 1997.

[52] Shervashidze, Nino, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt: *Weisfeiler-lehman graph kernels*. J. Mach. Learn. Res., 12(null):2539–2561, nov 2011, ISSN 1532-4435.

[53] Smaili, Fatima Zohra, Xin Gao, and Robert Hoehndorf: *Onto2vec: joint vector-based representation of biological entities and their ontology-based annotations*. Bioinformatics, 34(13):i52–i60, jun 2018. `https://doi.org/10.1093%2Fbioinformatics%2Fbty259`.

[54] Smaili, Fatima Zohra, Xin Gao, and Robert Hoehndorf: *Opa2vec: combining formal and informal content of biomedical ontologies to improve similarity-based prediction*, 2018. `https://arxiv.org/abs/1804.10922`.

[55] Soylu, Ahmet, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ernesto Jiménez-Ruiz, Martin Giese, Martin Skjæveland, Dag Hovland, Rudolf Schlatte, Sebastian Brandt, Hallstein Lie, and Ian Horrocks: *Optiquevqs: a visual query system over ontologies for industry*. Semantic Web, 9, August 2017.

[56] Sun, Chi, Xipeng Qiu, Yige Xu, and Xuanjing Huang: *How to fine-tune bert for text classification?*, 2019. `https://arxiv.org/abs/1905.05583`.

[57] Taylor, Wilson L.: *"cloze procedure": A new tool for measuring readability*. Journalism & Mass Communication Quarterly, 30:415 – 433, 1953.

[58] Team, Great Learning: *Types of neural networks and definition of neural network*. `"https://www.mygreatlearning.com/blog/types-of-neural-networks/"`, Accessed: 2022-02-15.

[59] Team, The Machine Learning: *Backpropagation deep learning*. `"https://the-learning-machine.com/article/dl/backpropagation?gclid=Cj0KCQiApL2QBhC8ARIsAGMm-KHk3f4FuVGXpjQr--dfdWpCc3glgE6-iYYb-lo_DGDJF2ovRFb26OsaArNwEALw_wcB"`, Accessed: 2022-02-15.

[60] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukasz Kaiser, and Illia Polosukhin: *Attention is all you need*. June 2017.

[61] Wang, Pu, Jian Hu, Hua Jun Zeng, and Zheng Chen: *Using wikipedia knowledge to improve text classification.* Knowl. Inf. Syst., 19:265–281, June 2009.

[62] Wikipedia contributors: *Semi-supervised learning — Wikipedia, the free encyclopedia.* https://en.wikipedia.org/w/index.php?title=Semi-supervised_learning&oldid=1049180726, 2021. [Online; accessed 11-March-2022].

[63] Wikipedia contributors: *Machine learning — Wikipedia, the free encyclopedia*, 2022. https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=1075772572, [Online; accessed 11-March-2022].

[64] Wikipedia contributors: *Natural language processing — Wikipedia, the free encyclopedia*, 2022. https://en.wikipedia.org/w/index.php?title=Natural_language_processing&oldid=1075824160, [Online; accessed 13-March-2022].

[65] Wikipedia contributors: *Perceptron — Wikipedia, the free encyclopedia*, 2022. https://en.wikipedia.org/w/index.php?title=Perceptron&oldid=1068824312, [Online; accessed 18-February-2022].

[66] Wikipedia contributors: *Transformer (machine learning model) — Wikipedia, the free encyclopedia*, 2022. https://en.wikipedia.org/w/index.php?title=Transformer_(machine_learning_model)&oldid=1070758033, [Online; accessed 15-February-2022].

[67] Yanardag, Pinar and S.V.N. Vishwanathan: *Deep graph kernels.* In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, page 1365–1374, New York, NY, USA, 2015. Association for Computing Machinery, ISBN 9781450336642. https://doi.org/10.1145/2783258.2783417.

[68] Zhang, Aston, Zachary C. Lipton, Mu Li, and Alexander J. Smola: *Dive into Deep Learning*. 2020. https://d2l.ai.

[69] Zhang, Zhengyan, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu: *ERNIE: Enhanced language representation with informative entities.* In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1441–1451, Florence, Italy, July 2019. Association for Computational Linguistics. https://aclanthology.org/P19-1139.