Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

Hierarchical Text Classification with Transformer-based Language Models

by
Jaco du Toit

Thesis presented in fulfilment of the requirements for the degree of Master of Science in the Faculty of Science at Stellenbosch University

Supervisor: Dr. Marcel Dunaiski

March 2024

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2024 ....

Abstract

Hierarchical Text Classification with Transformer-based Language Models

J.W. du Toit

Computer Science Division,
Department of Mathematical Sciences,
Stellenbosch University,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MSc (Computer Science)
March 2024

Hierarchical text classification (HTC) is a natural language processing (NLP) task which has the objective of classifying text documents into a set of classes from a structured class hierarchy. For example, news articles can be classified into a hierarchical class set which comprises broad categories such as “Politics” and “Sport” in higher-levels with associated finer-grained categories such as “Europe” and “Cycling” in lower-levels.

In recent years many different NLP approaches have been significantly improved through the use of transformer-based pre-trained language models (PLMs). PLMs are typically trained on large amounts of textual data through self-supervised tasks such that they acquire language understanding capabilities which can be used to solve various NLP tasks, including HTC. In this thesis, we propose three new approaches for leveraging transformer-based PLMs to improve classification performance on HTC tasks.

Our first approach formulates how hierarchy-aware prompts can be applied to discriminative language models such that it allows HTC tasks to scale to problems with very large hierarchical class structures. Our second approach uses label-wise attention mechanisms to obtain label-specific document representations which are used to fine-tune PLMs for HTC tasks. Furthermore, we propose a label-wise attention mechanism which splits the attention mechanisms into the different levels of the class hierarchy and leverages the predictions of all ancestor levels during the prediction of classes at a particular level. The third approach combines features extracted from a PLM and a topic model to train a classifier which comprises convolutional layers followed by a label-wise attention mechanism. We evaluate all three approaches comprehensively and show that our first two proposed approaches obtain state-of-the-art performances on three HTC benchmark datasets. Our results show that the use of prompts and label-wise attention mechanisms to fine-tune PLMs are very effective techniques for classifying text documents into hierarchical class sets. Furthermore, we show that these techniques are able to effectively leverage the language understanding capabilities of PLMs and incorporate the hierarchical class structure information to improve classification performance.

We also introduce three new HTC benchmark datasets which comprise the titles and abstracts of research publications from the Web of Science publication database with associated categories. The first two datasets use journal- and citation-based classification schemas respectively, while the third dataset combines these classifications with the aim of removing documents and classes which do not have a clear overlap between the two schemas. We show that this results in a more consistent classification of the publications. Finally, we perform experiments on these three datasets with the best-performing approaches proposed in this thesis to provide a baseline for future research.

Opsomming

Hiërargiese Teksklassifikasie met Transformatorgebaseerde Taalmodelle

J.W. du Toit

Afdeling Rekenaarwetenskap,
Departement van Wiskundige Wetenskappe,
Universiteit Stellenbosch,
Privaatsak X1, Matieland 7602, Suid-Afrika.

Tesis: MSc (Rekenaarwetenskap)

Maart 2024

Hiërargiese teksklassifikasie (HTC) is 'n natuurliketaalverwerkingstaak (NLP) wat die doel het om teksdokumente te klassifiseer in 'n klasversameling vanuit 'n gestruktureerde klashiërargie. Byvoorbeeld, nuusartikels kan geklassifiseer word in 'n hiërargiese klasversameling wat bestaan uit breë kategorieë soos "Politiek" en "Sport" op hoër vlakke met gepaardgaande fyner kategorieë soos "Europa" en "Fietsry" in laer vlakke.

In die afgelope paar jaar is baie verskillende metodes vir NLP-take aansienlik verbeter deur die gebruik van transformatorgebaseerde voorafgeleerde taalmodelle. Taalmodelle word tipies geleer op groot hoeveelhede teksdata deur middel van selftoesig take, sodat hulle taalbegripvermoëns verkry wat gebruik kan word om verskeie NLP-take op te los, insluitend HTC. In hierdie tesis stel ons drie nuwe metodes voor om transformatorgebaseerde taalmodelle te gebruik om klassifikasie vir HTC-take te verbeter.

Ons eerste metode formuleer hoe 'n hiërargie-bewuste por toegepas kan word op onderskeidmakende taalmodelle, sodat dit HTC-take toelaat om te skaal na probleme met baie groot hiërargiese klasstrukture. Ons tweede metode gebruik klasgewys aandagsmeganismes om klasspesifieke dokumentvoorstellings te verkry wat gebruik word om taalmodelle vir HTC-take te verfyn. Verder stel ons 'n klasgewys aandagsmeganisme voor wat die aandagsmeganismes in die verskillende vlakke van die klashiërargie verdeel en die voorspellings van alle hoër vlakke benut tydens die voorspelling van klasse op 'n spesifieke vlak. Die derde metode kombineer dokumentvoorstellings wat uit 'n taalmodel en 'n onderwerpmodel onttrek is om 'n klassifikasiemodel te leer wat konvolusielae bevat, gevolg deur 'n klasgewys aandagsmeganisme. Ons evalueer al drie metodes en wys dat ons eerste twee metodes beter presteer as onlangs voorgestelde metodes op drie HTC-maatstafdataversamelings. Ons resultate wys dat die gebruik van 'n hiërargie-bewuste por en klasgewys aandagsmeganismes om taalmodelle te verfyn baie effektiewe tegnieke is om teksdokumente in hiërargiese klasversamelings te klassifiseer. Verder wys ons dat hierdie tegnieke in staat is om die taalbegripvermoë van taalmodelle effektief te benut en die hiërargiese klasstruktuurinligting te gebruik om klassifikasie te verbeter.

Ons stel ook drie nuwe HTC-maatstafdataversamelings bekend wat die titels en opsommings van navorsingspublikasies uit die Web of Science publikasiedatabasis met gepaardgaande kategorieë bevat. Die eerste twee dataversamelings gebruik joernaalgebaseerde en aanhalinggebaseerde klassifikasieskema's onderskeidelik, terwyl die derde dataversameling hierdie klassifikasies kombineer met die doel om dokumente en klasse te verwyder wat nie 'n duidelike oorvleueling tussen die twee skemas het nie. Ons wys dat ons voorgestelde metode lei tot 'n meer konsekwente klassifikasie van die publikasies. Laastens evalueer ons die bes-presterende metodes wat in hierdie tesis voorgestel is op die drie nuwe dataversamelings om 'n basislyn vir toekomstige navorsing te verskaf.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

AN	Artificial Neuron
ANN	Artificial Neural Network
BERT	Bidirectional Encoder Representations from Transformers
BOW	Bag Of Words
BPTT	Backpropagation Through Time
CAML	Convolutional Attention for Multi-Label classification
CBOW	Continuous Bag Of Words
CNN	Convolutional Neural Network
CREST	Centre for Research on Evaluation, Science and Technology
CT	Citation Topics
DeBERTa	Decoding-enhanced BERT with disentangled attention
DPA	Dot Product Attention
DPT	Prompt Tuning framework for Discriminative language models
DR-CAML	Description Regularised CAML
ELECTRA	Efficiently Learning an Encoder that Classifies Token Replacements Accurately
FCL	Fully-Connected Layer
FFNN	Feed-Forward Neural Network
GA	General Attention
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GHLA	Global Hierarchical Label-wise Attention
GloVe	Global Vectors
GNN	Graph Neural Network
GRU	Gated Recurrent Unit network
HA-GRU	Gated Recurrent Unit with a Hierarchical Attention mechanism

HBGL	Hierarchy-guided BERT with Global and Local hierarchies
HDLTex	Hierarchical Deep Learning for Text Classification
HE-HMTC	Hybrid Embedding-based text representation for Hierarchical Multi-label Text Classification
HFT-CNN	Hierarchical Fine-Tuning based Convolutional Neural Network
HGCLR	Hierarchy Guided Contrastive Learning
HiAGM	Hierarchy-Aware Global Model
HiAGM-LA	HiAGM Multi-label Attention
HiAGM-TP	HiAGM Text feature Propagation
HiLAT	Hierarchical Label-wise Attention Transformer
HiMatch	Hierarchy-aware label semantics Matching network
HLA	Hierarchical Label-wise Attention
HPT	Hierarchy-aware Prompt Tuning
HPTD	Hierarchy-aware Prompt Tuning for Discriminative language models
HTC	Hierarchical Text Classification
HTrans	Hierarchical Transfer Learning
ICD	International Classification of Diseases
JT	Journal-based Topics
JTF	Journal-based Topics Filtered
LAAT	Label Attention Model
LCN	Local Classifier per Node
LCPN	Local Classifier per Parent Node
LCL	Local Classifier per Level
LDA	Latent Dirichlet Allocation
LHTE	Local Hierarchy-aware Text Encoder
LSAN	Label Specific Attention Network
LSTM	Long Short-Term Memory network
MAGNET	Multi-label Text classification using Attention based Graph Neural NETwork
MLC	Multi-Label Classification
MLM	Masked Language Modelling
MLTC	Multi-Label Text Classification
MultiResCNN	Multi-Filter Residual Convolutional Neural Network
M-XLNet	Multilabel XLNet

NLP	Natural Language Processing
NSP	Next Sentence Prediction
NYT	New York Times
PAAMHiA-T5	Hierarchy-Aware T5 model with Path-Adaptive Attention Mechanism
PLM	Pre-trained Language Model
RCV1-V2	Reuters Corpus Volume 1 Version 2
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RoBERTa	Robustly optimised BERT pre-training approach
RTD	Replaced Token Detection
TF-IDF	Term Frequency-Inverse Document Frequency
TIS	Topic Identification System
TopAttCNN	Topic Attention Convolutional Neural Network
WOS	Web Of Science

Chapter 1

Introduction

Text classification is a common task in the field of natural language processing (NLP) with the objective of classifying text documents into a set of classes such that the documents can be categorised for improved organisation and navigation. However, standard text classification approaches often use categories which are too broad or too vague, making them unsuitable for navigating and filtering through a large collection of documents.

Hierarchical text classification (HTC) is a form of text classification which aims to classify text documents into classes from a predefined structured class hierarchy. Figure 1.1 depicts an example of a class hierarchy for news articles which shows how the level of granularity increases from the highest to the lowest layers of the hierarchy. Therefore, a HTC system can be used to improve the organisation and navigation of a large document collection by reducing the search scope from a large number of documents to a smaller subset of documents with detailed categories.

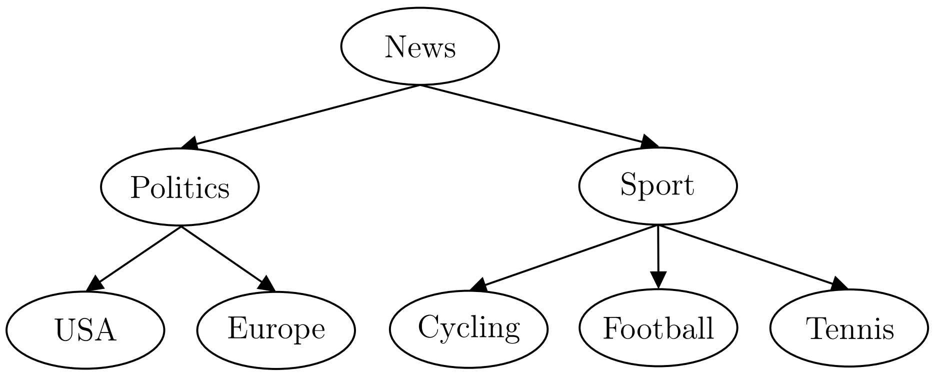


Figure 1.1: Example of hierarchical class structure for news articles.

Many HTC approaches have been proposed which attempt to leverage the class hierarchy information in various ways to improve classification performance. However, using transformer-based [84] pre-trained language models (PLMs) [14, 17, 66, 101] has become the standard approach for most NLP tasks in recent years. This is due to the language understanding capabilities obtained by the PLM during pre-training which can be leveraged in various ways to perform downstream NLP tasks effectively. Therefore, recent approaches for solving HTC tasks combine representations of the hierarchical class structure with the language understanding capabilities of transformer-based PLMs to improve performance on HTC tasks [31, 32, 89, 90].

In this thesis we investigate the use of transformer-based PLMs for HTC tasks as well as effective ways of incorporating the hierarchical class structure information into the classification process to improve performance on these tasks.

1.1 Motivation

Over the past few decades there has been a rapid growth in the number of available digital text documents in various domains such as news, research, literature, and social media. However, the abundance of textual data often leads to an overload of information, which prevents users from efficiently navigating through large document collections and long lists of search results to find relevant information in their fields of interest. Therefore, users waste time and effort while looking for relevant documents due to the typical unstructured format of textual information.

Categorising these documents into a detailed set of classes which summarise their content can be beneficial for improving the accessibility of information to users since it allows them to navigate and filter through a large collection of documents more efficiently. However, manually categorising large document collections is expensive due to the amount of human effort and expertise required to accurately categorise documents. Therefore, creating a system that can automatically and accurately categorise text documents into a set of classes without the intervention of humans is an extremely valuable tool. Text classification approaches have the objective of assigning a text document to one or more classes from a predefined class set. However, these approaches typically use very general or high-level categories such that users are often still required to navigate through a large number of documents when these categories are used as filters. For example, these approaches may categorise news articles into broad classes such as “Politics”, “Business”, and “Sport”.

Fortunately, text documents can often be categorised into a set of classes with a hierarchical structure, where higher-level classes comprise broad categories while more detailed categories are used for the lower levels of the hierarchy. For example, the broad news article classes mentioned above (“Politics”, “Business”, and “Sport”) can have associated finer-grained categories such as “Europe”, “Financial Markets” and “Cycling” respectively. These hierarchical class structures are often very useful for the organisation of text documents since they allow users to reduce their search scope from a large collection of documents to a smaller subset of documents with finer-grained categories. Furthermore, the hierarchical class structure allows users to select the level of granularity that they prefer when applying the categories as filters in a collection of documents.

HTC systems have the objective of assigning text documents to the classes from one or more paths in the structured class hierarchy. Due to the value that HTC systems can provide, various approaches have been proposed which aim to leverage the hierarchical class structure to improve performance on HTC tasks. Recent HTC approaches have shown that leveraging the language understanding capabilities of transformer-based PLMs leads to significant performance improvements on HTC tasks [31, 32, 89, 90]. Moreover, these approaches use varying techniques to combine the hierarchical information of the class taxonomy with PLMs to improve classification performance.

Although recent approaches achieve impressive results on benchmark HTC datasets [31, 32, 89, 90], several areas of research remain unexplored for HTC tasks. More specifically, several advancements in standard "flat" text classification tasks, which do not have a hierarchical class structure, have not been comprehensively investigated for HTC tasks and can be exploited to improve classification performance. Furthermore, it is still unclear what the best approaches are for incorporating the class hierarchy information into the classification procedure. Particularly, combining this information with the language understanding capabilities of a PLM is a difficult problem which requires further research and is one of the areas explored in this thesis.

The main motivation for the research presented in this thesis is to investigate unexplored areas of research for leveraging the language understanding capabilities of transformer-based PLMs to solve HTC tasks. On a higher level, the motivation behind studying effective ways of using PLMs on the downstream task of HTC is that these models will likely improve over time as has been shown with the introduction of new transformer-based PLMs over the past few years [14, 17, 26, 27, 101]. Therefore, by focusing on how to leverage PLMs, we may gain insights into how these models behave on HTC tasks and apply these insights to improve classification performance as the capabilities of PLMs improve over time.

1.2 Objectives

The main objective of the research presented in this thesis is to investigate the use of transformer-based PLMs for HTC tasks and how to incorporate the hierarchical class structure information into the classification process to improve performance. Working towards this objective, we have identified the following sub-objectives:

- Review recent HTC approaches which use transformer-based PLMs.

- Identify possible shortcomings of proposed approaches and promising unexplored areas of research.
- Identify advancements in standard text classification approaches which have not been applied to HTC tasks.
- Propose new HTC approaches from the identified unexplored areas of research.
- Create new benchmark HTC datasets to enable more reliable evaluations and comparisons between the performance of different HTC approaches.

1.3 Thesis Overview

Chapter 2 provides background information on the main concepts required to follow the remainder of this thesis.

Chapter 3 provides a literature review which covers the previously proposed HTC approaches that are most relevant to this thesis.

Chapters 4, 5, 6, and 7 are each structured as individual papers comprising an introduction, a detailed background section for the specific chapter, as well as their self-contained methodology, experiments, and conclusion sections. Chapters 4, 5, and 6 each propose a new approach for solving HTC tasks while Chapter 7 introduces three new HTC benchmark datasets.

Chapter 4 proposes the Hierarchy-aware Prompt Tuning for Discriminative language models (HPTD) approach which applies the prompt tuning paradigm to discriminative language models to solve HTC tasks.

Chapter 5 proposes an approach which uses label-wise attention mechanisms to fine-tune transformer-based PLMs for HTC tasks.

Chapter 6 proposes a HTC approach that uses a PLM and a topic model to extract features from text documents which are used to train a classifier that comprises convolutional layers and a label-wise attention mechanism.

Chapter 7 describes the methodology used to create three new benchmark HTC datasets which comprise the titles and abstracts of research publications and their associated classifications. These datasets are subsequently used to evaluate the performance of the best-performing approaches proposed in this thesis.

Chapter 8 concludes the thesis with a discussion of the main results, limitations, and suggestions for future research.

Chapter 2

Background

2.1 Machine Learning

Machine learning is broadly defined as the ability of an algorithm to learn and improve on a particular task as it is presented with more data or information over time, without being explicitly programmed to do so. Machine learning algorithms are typically grouped into three categories: supervised, unsupervised, and reinforcement learning. Unsupervised learning aims to learn patterns, relationships, or structures in unlabelled data, i.e., inputs that do not have associated outputs. In reinforcement learning, the algorithm interacts with an environment and receives feedback in the form of rewards and penalties which drives the learning process. Supervised learning uses a training set of example inputs along with their true outputs and has the objective of learning a functional mapping from the input to the output space. We focus on the supervised learning paradigm in this thesis.

Supervised learning techniques can differ with regard to the type or structure of their input and output data. The inputs to supervised learning algorithms commonly include textual data, image data, or numerical data from which features are extracted that are used by the learning algorithms. These algorithms are generally used to output real numbers (regression) or assign labels to an example input (classification).

In this thesis we use supervised machine learning techniques for classification tasks in the domain of natural language text. Specifically, we focus on artificial neural network (ANN) architectures, which are a class of machine learning algorithms inspired by the structure and interaction of neurons in the brain [55, 71]. The feed-forward neural network (FFNN) is considered the foundational ANN architecture since it forms the basis of most other ANN architectures [29, 45, 72].

A FFNN consists of units (or nodes) called artificial neurons (ANs) and connections between them. An AN receives input signals from other ANs in the network and these input signals are grouped into a vector $\mathbf{x} = [x_1, \ldots, x_I] \in \mathbb{R}^I$, where I is the number of input signals. The input signals have associated weights that represent the importance of each input signal and are represented as $\mathbf{w} = [w_1, \ldots, w_I] \in \mathbb{R}^I$. These weights are randomly initialised and learnt during training through the backpropagation and gradient descent algorithms described below. The AN uses the weighted input signals to perform a functional mapping and obtain an output signal. The output signal is most commonly calculated by obtaining the weighted sum of the input signals, adding a term known as the bias unit (b), and passing the sum through a function (f) known as the activation function. Therefore, the output signal of the AN is calculated as:

$$f\left((\sum_{i=1}^{I} w_i x_i) + b \right) = f(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.1.1}$$

The activation function determines the effect of an AN and non-linear activation functions allow ANNs to learn more complex functional mappings from the input space to the output space. The most commonly used activation functions include the rectified linear unit (ReLU), sigmoid (σ), and tanh functions which are calculated as:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.1.2}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1.3}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.1.4}$$

A standard FFNN is broadly divided into an input layer, hidden layer, and output layer as depicted in Figure 2.1. The input layer consists of a number of nodes which represent the input to the network, where the structure of the input varies for different machine learning tasks. These input nodes capture different characteristics or features of an input data instance. We represent the values of the input nodes with the vector $\mathbf{x} = [x_1, \ldots, x_I] \in \mathbb{R}^I$, where I is again the number of input nodes. The hidden layer consists of a number of nodes which are connected to each of the input nodes separately with associated weights. Each hidden node uses the input signals and weights to obtain the output signal as shown in Equation (2.1.1). When two layers are connected in this manner it is referred to as a fully-connected layer (FCL). The weights of the FCL that connect the input and hidden layer nodes are represented by a weight matrix $\mathbf{W}^{(h)} = [\mathbf{w}_1^{(h)}, \ldots, \mathbf{w}_H^{(h)}] \in \mathbb{R}^{H \times I}$, where H is the number of hidden nodes and $\mathbf{w}_i^{(h)} \forall\, i \in \{1, \ldots, H\}$ is the weight vector for the input signals of the i-th hidden node. Therefore, $\mathbf{w}_i^{(h)} = [w_{i,1}^{(h)}, \ldots, w_{i,I}^{(h)}] \in \mathbb{R}^I$ where $w_{i,j}^{(h)} \forall\, j \in \{1, \ldots, I\}$ is the weight of the j-th input node for the i-th hidden node. The bias units are also combined into a vector $\mathbf{b}^{(h)} = [b_1^{(h)}, \ldots, b_H^{(h)}] \in \mathbb{R}^H$ such that the outputs for all hidden nodes are calculated as:

$$\mathbf{h} = f(\mathbf{W}^{(h)}\mathbf{x} + \mathbf{b}^{(h)}) \tag{2.1.5}$$

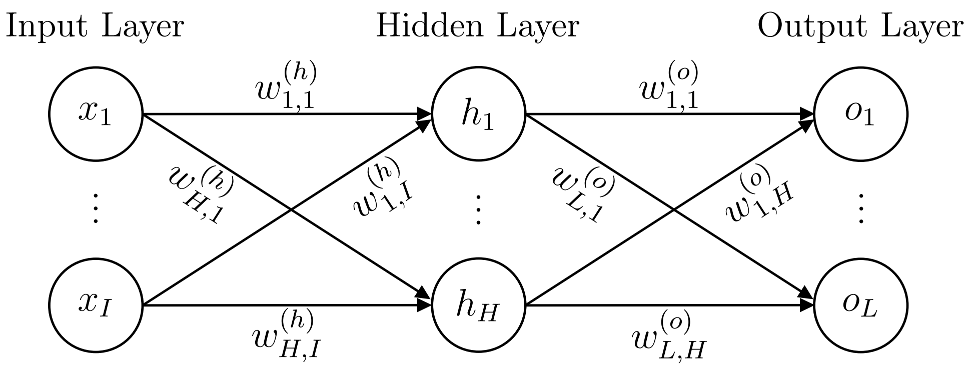


Figure 2.1: Standard feed-forward neural network architecture. The input node values $(x_1, \ldots, x_I)$ are passed with associated weights $(w^{(h)}_{1,1}, \ldots, w^{(h)}_{H,1}, \ldots, w^{(h)}_{1,I}, \ldots, w^{(h)}_{H,I})$ to obtain the hidden node values $(h_1, \ldots, h_H)$. Similarly, the hidden node values are passed with associated weights to obtain the values of the output nodes $(o_1, \ldots, o_L)$ which form the output of the network. We omit the bias units for sake of clarity.

where $\mathbf{h} = [h_1, \ldots, h_H] \in \mathbb{R}^H$ represents the output values for each hidden node. Similarly, the hidden node output signals are passed with associated weights and biases to the output layer which consists of one or more output nodes representing the output of the network. Suppose the output layer comprises L nodes, the output of the network is calculated as:

$$\mathbf{o} = f(\mathbf{W}^{(o)}\mathbf{h} + \mathbf{b}^{(o)}) \tag{2.1.6}$$

where $\mathbf{o} = [o_1, \ldots, o_L] \in \mathbb{R}^L$ represents the values of each output node, and $\mathbf{W}^{(o)} \in \mathbb{R}^{L \times H}$ and $\mathbf{b}^{(o)} \in \mathbb{R}^L$ are the associated weights and biases connecting the hidden and output nodes.

The values of the output nodes are used in an error (or loss) function which measures the difference between the predicted output of the model and the expected output. The calculated error ($\mathcal{L}$) is used to drive the learning process of the model parameters (weights and biases) for each layer during training. The parameters are typically learnt through two algorithms: backpropagation and a gradient descent-based optimisation algorithm. These algorithms are used by the ANN to learn suitable weights and biases such that the model represents a functional mapping from the input to the output space based on the training data.

The backpropagation algorithm [48] propagates the error of the output nodes back through the layers of the network to calculate the gradients of the error function with respect to each of the parameters in the network. These gradients are used by a gradient descent-based optimisation algorithm [69] to adjust the model parameters such that the model moves to a lower point in the error function landscape and thereby reduces the error of the network. The standard gradient descent algorithm updates a particular parameter w as follows:

$$w = w - \epsilon \frac{\partial \mathcal{L}}{\partial w} \tag{2.1.7}$$

where ϵ is the learning rate which controls the size of the parameter updates and $\frac{\partial \mathcal{L}}{\partial w}$ is the gradient of the error with respect to w as determined by the backpropagation algorithm.

In the standard gradient descent algorithm, the entire training dataset is used to determine the error of the network by accumulating the outputs of the error function for each instance. Therefore, the error of the entire training dataset is used to perform the update on the parameters.

The stochastic gradient descent algorithm [69] modifies standard gradient descent by accumulating the errors of a randomly selected subset of training instances known as a mini-batch. Therefore, each update step only considers a batch of training instances at a time such that the model performs more frequent update steps and therefore converges faster [69]. The batches are randomly sampled without replacement until all of the instances in the training set have been used to update the parameters. An iteration of the entire training set passing through the network and updating the parameters is known as an epoch.

2.2 Classification Tasks

Classification is a machine learning task which has the objective of classifying a particular data instance, such as a text document or image, to belong to one or more classes (or labels) from a predefined set of classes. Single-label classification tasks aim to assign an instance to a single class from a set of classes $C = \{c_1, \ldots, c_L\}$, where L is the number of classes. A single-label classification task is termed a binary classification task if $L = 2$ and a multi-class classification task if $L > 2$. In contrast to single-label classification, multi-label classification (MLC) allows each instance to be associated with one or more classes from the set of classes C.

Suppose we have an input feature vector space $\mathbf{X}$ where the feature vector representation of the i-th instance in a dataset is given as $\mathbf{x}_i \in \mathbf{X}$ and its associated class output is given as $\mathbf{y}_i \subseteq C$. The training dataset is defined as $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) | 1 \leq i \leq N\}$ where N is the number of instances in the training set. The goal of MLC problems is to learn a functional mapping $f : \mathbf{X} \to 2^C$ from the training set $\mathcal{D}$, where 2^C represents the power set of C, i.e., the set of all possible subsets of C. Therefore, a classifier can use this functional mapping to predict the classes of an instance as $Y' \subseteq C$.

In this thesis we focus on text classification which has the objective of classifying text documents into classes from a predefined class set. Text classification may be used to solve various tasks such as:

- Topic labelling, where text documents such as news articles or scientific papers are categorised by assigning them to relevant categories.
- Sentiment analysis, which analyses people's opinions in textual data such as movie reviews or social media posts and determines their tone or polarity.
- Named entity recognition, which locates and classifies named entities such as persons, locations, and organisations in a text document.
- Question answering, where the objective is to select suitable answers to a question from a set of potential answers.
- Language detection, which detects the language of a text document.

2.3 Hierarchical Classification

Hierarchical classification is an extension of MLC where the class set is structured in a hierarchical taxonomy. The hierarchical class set is typically structured such that high-level classes are more broad while lower-level classes provide more detailed classifications.

Hierarchical classification tasks define the structured class hierarchy as a directed acyclic graph $\mathcal{H} = (C, E)$, where C is the set of class nodes and E is the set of all edges that form the hierarchical structure's parent-child relationships. The objective of hierarchical classification tasks is to classify an instance as belonging to a class set $Y' \subseteq C$. In this thesis, we assume that each node, apart from the root node, has only one parent node such that the graph forms a tree structure. Therefore, the class set Y' comprises one or more paths in the class hierarchy $\mathcal{H}$.

Hierarchical classification approaches are broadly categorised into three groups: flat, local, and global approaches [78]. We describe these three approaches and provide the framework for hierarchical classification tasks and algorithms as proposed by Silla and Freitas [78].

2.3.1 Flat Classifier Approaches

Silla and Freitas [78] describe the flat classifier approach as the simplest strategy for solving hierarchical classification problems since it ignores the class hierarchy and only considers leaf-node classes. This approach solves the hierarchical classification problem by assuming that when an instance is assigned to a leaf node, all of the associated ancestor nodes are assigned to that instance as well. Flat classification approaches have the disadvantage that they require a multi-label classifier to distinguish between all of the leaf nodes in the class hierarchy without leveraging the useful relationships between the nodes in higher levels. Furthermore, this approach is unable to handle non-mandatory leaf node problems where instances are not required to be assigned to leaf nodes of the class hierarchy but may be assigned directly to internal nodes as well.

In this thesis, we do not consider this type of flat classification approach due to the mandatory leaf node requirement. However, we consider a different type of flat classification approach where the class hierarchy is ignored and internal nodes are treated as leaf nodes, thereby “flattening” the hierarchical class structure. This approach transforms the classification task into a MLC task. For the remainder of this thesis, we use the term “flat classification” to refer to this type of classification approach. This approach has two major disadvantages. Firstly, it ignores the useful relationships between nodes in the class hierarchy. And secondly, it can introduce inconsistent class assignments, where instances are associated with classes from child and ancestor nodes that have no relation in the class hierarchy.

2.3.2 Local Classifier Approaches

Local classifier approaches take into account the hierarchical structure of the class taxonomy and combine the results from multiple classifiers to solve hierarchical classification problems [78]. The most common local classifier approaches are the local classifier per node (LCN), per parent node (LCPN), and per level (LCL) approaches. These three approaches differ in the way the classifiers are trained, but use similar top-down evaluation steps where the algorithm first predicts the top-level and uses the predicted classes to narrow the choices of the classes to predict in the lower levels. Therefore, these approaches have the disadvantage that an incorrectly predicted class at a certain level of the hierarchy will lead to incorrect predictions for that instance in all of the lower levels.

The LCN approach trains a binary classifier for each non-root node in the class hierarchy, where the classifier of a node predicts whether an instance belongs to that class or not. The LCPN approach trains a classifier for each non-leaf node in the class hierarchy, where a parent node classifier is used to distinguish between the classes of its child nodes. The LCL approach trains a classifier for each level of the class hierarchy and each classifier predicts the classes to which the instance belongs in the associated level. Figures 2.2, 2.3, and 2.4 depict the classifiers that are constructed for the LCN, LCPN, and LCL approaches respectively.

2.3.3 Global Classifier Approaches

Global classifier approaches build a single classification model which considers the overall hierarchical class structure during training and inference as shown in Figure 2.5. Silla and Freitas [78] state that although there is no specific main characteristic that is shared by all global classifier approaches,

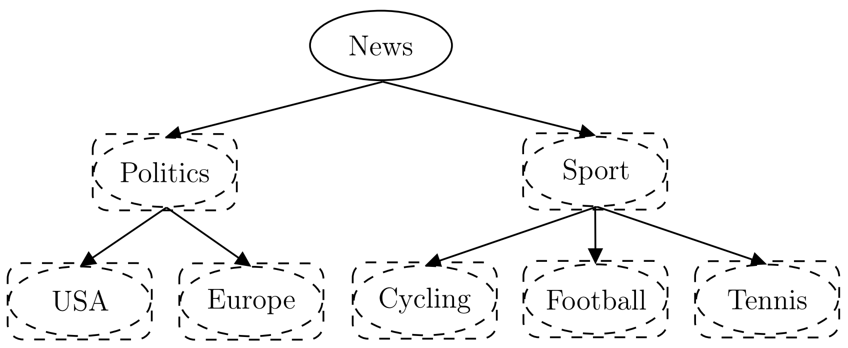


Figure 2.2: Local classifier per node approach, where dashed boxes represent binary classifiers and dashed ovals represent the class that they are predicting an instance to belong to or not. For example, the binary classifier for Politics predicts whether an instance belongs to the Politics class.

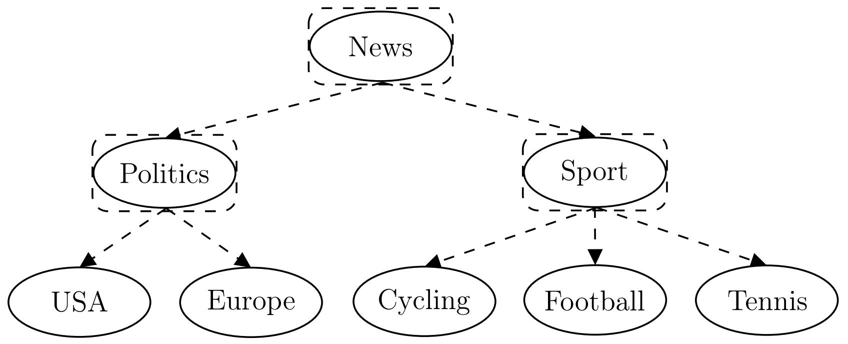


Figure 2.3: Local classifier per parent node approach, where dashed boxes represent classifiers and dashed arrows represent the classes that the classifier is trying to predict. For example, the classifier for Politics predicts whether an instance belongs to the USA or Europe class.

they all take into account the whole class hierarchy at once and do not have the modular local training found in local classifier approaches. Several global classifier approaches have been proposed with significantly different strategies for capturing the hierarchical class structure information in a single model. We discuss several of these approaches in Chapter 3.

2.3.4 Framework for Hierarchical Classification

Silla and Freitas [78] proposed a unifying framework for describing the characteristics of hierarchical classification problems and algorithms. This allows for an effective categorisation of different hierarchical classification tasks and algorithms used to solve such tasks.

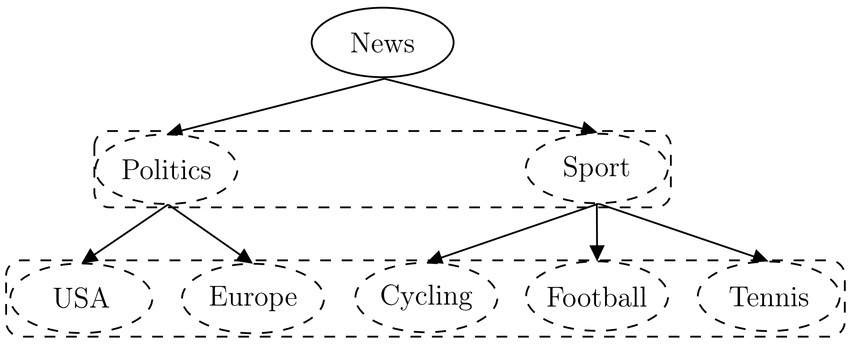


Figure 2.4: Local classifier per level approach, where dashed boxes represent classifiers and dashed ovals within a box represent the classes that the classifier is trying to predict. For example, the final-level classifier predicts whether an instance belongs to the USA, Europe, Cycling, Football, or Tennis class.

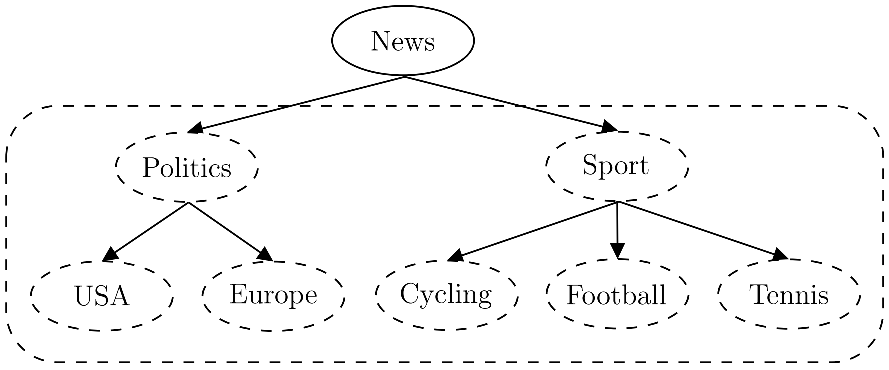


Figure 2.5: Global classifier approach, where the dashed box represents the classifier and dashed ovals represent the classes that the classifier is trying to predict.

2.3.4.1 Framework for Hierarchical Classification Problems

Silla and Freitas [78] propose that hierarchical classification algorithms can be described as a 3-tuple $\langle \Upsilon, \Psi, \Phi \rangle$, where:

- Υ describes the graph structure of the hierarchical classes (nodes) and their relationships (edges). Therefore, the values for this attribute can be:
  - D (DAG), which indicates that the classes form a Directed Acyclic Graph (DAG) structure. DAGs are graph structures which do not allow connections between nodes to form loops, i.e., there is no path of connections that leads from a node back to itself.
  - T (Tree), which indicates that the classes form a tree structure. Tree graph structures are DAGs with the restriction that a child node can only have one parent.
- Ψ describes whether an instance may have labels that are associated with

a single or multiple paths in the class hierarchy. Therefore, the values for this attribute can be one of the following:

  - SPL (Single Path of Labels), which implies that an instance may only have one class per level.
  - MPL (Multiple Paths of Labels), which implies that an instance may have multiple classes per level.

- Φ describes the label depth of instances in the dataset. The values for this attribute can be one of the following:
  - FD (Full Depth labelling), which indicates that all instances are assigned at least one class at each level of the hierarchy.
  - PD (Partial Depth labelling), which indicates that at least one instance has a partial depth labelling such that the class at a certain level is unknown, i.e., the instance is only labelled up until a certain level.

2.3.4.2 Framework for Hierarchical Classification Algorithms

Silla and Freitas [78] propose that hierarchical classification algorithms can be described as a 4-tuple $\langle \Delta, \Xi, \Omega, \Theta \rangle$, where:

- Δ indicates whether the algorithm can predict labels in just one or multiple different paths. Therefore, the value for this attribute can be:
  - SPP (Single Path Prediction), which indicates that the algorithm can assign only one path of class labels to an instance.
  - MPP (Multiple Path Prediction), which indicates that the algorithm can assign multiple paths of class labels to an instance.
- Ξ describes the prediction depth of the algorithm which can take on two values:
  - MLNP (Mandatory Leaf-Node Prediction), which indicates that the algorithm always assigns classes from each level up until the leaf classes to an instance.
  - NMLNP (Non-Mandatory Leaf-Node Prediction), which indicates that the algorithm can assign classes at any level, but is not required to assign classes from each level to each instance.
- Ω is the taxonomy structure that the algorithm is able to handle which can take on two values:
  - D (DAG), which indicates that the predicted classes form a DAG structure.

  - T (Tree), which indicates that the predicted classes form a tree structure.

- Θ is the categorisation of the algorithm under the proposed taxonomy which can take on four values:
  - LCN (Local Classifier per Node)
  - LCPN (Local Classifier per Parent Node)
  - LCL (Local Classifier per Level)
  - GC (Global Classifier)

2.4 Feature Extraction

Feature extraction in text classification tasks is the process of converting raw text to a numerical representation which can be used in a machine learning algorithm. Therefore, the objective is to obtain a numerical representation of the text document which captures the distinctive attributes of that document and thereby enables a classification model to form a functional mapping from the input space to the output space.

The raw text is first divided into units called tokens. These units are combined to form a sequence of tokens that represent the original text. In earlier approaches, these tokens were typically formed by semantic words. However, the approaches considered in this thesis form tokens as a combination of words and parts of words (or subwords) by determining the most frequent subwords in a large training corpus [42, 93].

The sequences of tokens are used in various ways to obtain a numerical representation of text documents. Early methods for text feature extraction include one-hot encoding, bag of words (BOW), and term frequency-inverse document frequency (TF-IDF). More advanced methods such as word2vec [58] and global vectors for word representation (GloVe) [63] have been proposed which create vector representations for each token. However, these approaches have also generally been replaced by transformer-based PLMs [14, 17, 101] which we discuss in Section 2.6.

2.4.1 One-hot-encoding

One-hot-encoding is a simple feature extraction technique which transforms each document into a V-dimensional vector, where V is the size of the vocabulary, and the vocabulary is the set of unique words in the corpus of documents. Each element in the vector representation of a document represents a certain word and whether it is contained in the document (encoded with 1) or not (encoded with 0). Therefore, suppose we have two simple sentences “the cat and the dog” and “the tree in the forest” so that the vocabulary is given as

$\mathcal{V}$ = {"the", "cat", "and", "dog", "tree", "in", "forest"}. One-hot-encoding represents the sentences as:

$$\text{"the cat and the dog"} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$
$$\text{"the tree in the forest"} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Therefore, for a larger vocabulary this approach has several disadvantages such as creating a very sparse vector representation of the document and not being able to handle words that do not appear in the training data.

2.4.2 Bag of Words

The BOW approach is similar to one-hot-encoding, but instead of each element in the vector representing whether a word occurs or not, the element represents the number of times that the associated word occurs in the document. Therefore, the BOW approach represents the example sentences as:

$$\text{"the cat and the dog"} = \begin{bmatrix} 2 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$
$$\text{"the tree in the forest"} = \begin{bmatrix} 2 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

This approach usually ignores words that are uncommon by removing words that occur less than a certain number of times in order to reduce the sparsity of the vector representation. However, the BOW approach typically still results in sparse and high-dimensional vector representations for large vocabularies.

2.4.3 TF-IDF

While the BOW approach assigns higher values to more frequent words, TF-IDF incorporates the idea that some words that occur less frequently may provide more relevant information for representing a document. Therefore, TF-IDF combines the Term Frequency (TF) and Inverse Document Frequency (IDF) measures to represent a document as a distribution over the words in the vocabulary where more relevant words are assigned higher weights.

TF measures the number times a term t appears in a document d, and is typically normalised by the total number of words in the document such that it is calculated as:

$$\text{TF}(t,d) = \frac{freq(t,d)}{\sum_{t' \in d} freq(t',d)} \tag{2.4.1}$$

where $freq(t,d)$ is the number of times term t occurs in document d.

IDF measures how rare a word is in the corpus of documents (D) and is typically calculated as:

$$\text{IDF}(t, D) = \log \frac{|D|}{df(t, D)} \tag{2.4.2}$$

where $|D|$ is the number of documents in the corpus, and $df(t, D)$ is the number of documents in D which contain the term t.

The TF-IDF weight for a term t in a document d is therefore calculated as:

$$\text{TF-IDF}(t, d, D) = \text{TF}(t, d) \cdot \text{IDF}(t, D) \tag{2.4.3}$$

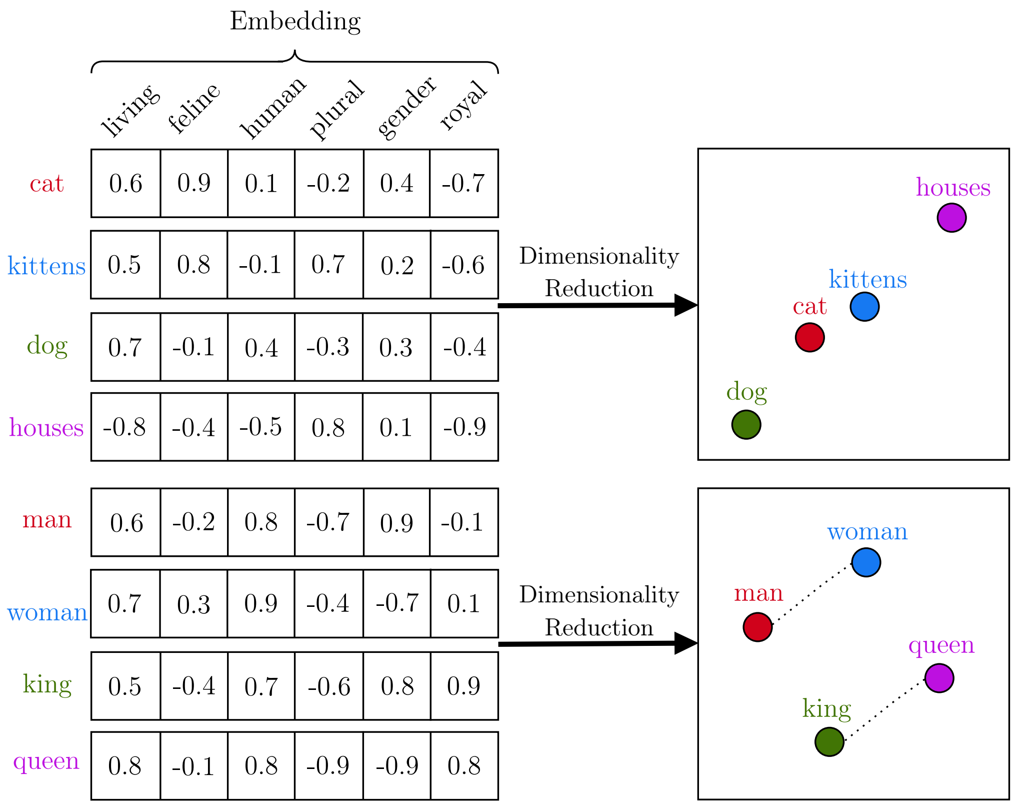
2.4.4 Word Embeddings

The feature extraction approaches described above have several disadvantages which include: high-dimensional and sparse vector representations, and the failure to capture information relating to similarities between words and the context in which words occur. More recent approaches obtain vector representations for each token in a corpus of documents which are referred to as word embeddings [5, 58, 63]. These word embeddings aim to capture different characteristics of the words in a document such as the semantic and contextual meaning such that similar words are close in the embedding space while unrelated words have a greater distance between them.

In Figure 2.6 we present an example of word embeddings where each element of the embedding represents a characteristic. The example also shows the distance between the different word embeddings when mapped to 2-dimensional vectors for visualisation. For example, the distance between the word embeddings for "man" and "woman" is approximately the same as the distance between "king" and "queen". Therefore, we can determine the approximate word embedding for "queen" as:

$$\text{“king”} - \text{“man”} + \text{“woman”} \approx \text{“queen”} \tag{2.4.4}$$

Bengio *et al.* [5] proposed an approach which uses a FFNN to learn word embeddings for the different words in a corpus of documents. Their approach used a language modelling task which has the objective of predicting the next word given the context within which a word occurs. This is done by using the previous words in the input sequence as context and predicting a probability distribution for the next word over all the words in the vocabulary. Each word in the vocabulary is represented by a vector which is randomly initialised and learnt during training. When the t-th word in the text sequence is predicted, the model passes the word embeddings for all the words before word t through a FFNN which obtains the output as a probability distribution over all of the words in the vocabulary. Therefore, the model uses the idea that words that occur in similar contexts should have a similar semantic meaning and therefore



	living	feline	human	plural	gender	royal
cat	0.6	0.9	0.1	-0.2	0.4	-0.7
kittens	0.5	0.8	-0.1	0.7	0.2	-0.6
dog	0.7	-0.1	0.4	-0.3	0.3	-0.4
houses	-0.8	-0.4	-0.5	0.8	0.1	-0.9
man	0.6	-0.2	0.8	-0.7	0.9	-0.1
woman	0.7	0.3	0.9	-0.4	-0.7	0.1
king	0.5	-0.4	0.7	-0.6	0.8	0.9
queen	0.8	-0.1	0.8	-0.9	-0.9	0.8

Figure 2.6: Visualisation of word embeddings, where each element of the vector captures a characteristic of the word. These embeddings are mapped to a 2-dimensional representation and visualised in the boxes on the right.

have similar word embeddings. This approach showed that ANNs can be used to obtain vector representations that capture the semantics and context of words.

Word2vec [58] is a another feature extraction technique which uses ANNs to obtain word embeddings for the words in a corpus of documents. The word2vec methods include the continuous bag of words (CBOW) and skip-gram algorithms [58]. CBOW has the objective of obtaining word embeddings for a particular word by simply averaging the embeddings of surrounding words. As opposed to the language modelling task proposed by Bengio *et al.* [5] which tries to predict the next word by looking at the previous words, CBOW tries to predict the current word by looking at all of the surrounding words. It is called the CBOW algorithm because it ignores the position of words such as in the BOW approach. On the other hand, the skip-gram algorithm tries to predict the surrounding words of a given word. This is done through the creation of positive and negative input samples which contain the current word and either correct or randomly sampled context words for the positive and negative samples respectively. The skip-gram model uses the word embeddings of the current word and context words to predict whether it is a positive or negative sample. This strategy allows the model to obtain similar embeddings for semantically similar words by being aware of contextually relevant words. GloVe [63] is another text feature extraction approach which improves the word2vec approach by incorporating count-based methods which capture global statistics such as the co-occurrences of words in the corpus of documents.

2.5 Deep Learning Architectures

According to LeCun *et al.* [44] "representation learning is a set of methods that allow a machine to be fed with raw data and to automatically discover the representations needed for detection or classification". Deep learning methods are a class of representation learning methods which obtain multiple levels of representations by stacking multiple non-linear layers that transform each representation into a representation at a higher, more abstract level [44]. Therefore, deep learning architectures typically do not require the manual selection and engineering of features since they learn to find and use the most relevant features from the raw input data for a specific task.

Several deep learning ANN architectures have been proposed of which the ones most relevant to this thesis include: the convolutional neural network (CNN) [35, 45], the recurrent neural network (RNN) [13, 28, 29], and the transformer model [84]. In this section we discuss these deep learning architectures with reference to their application in NLP tasks.

2.5.1 Convolutional Neural Networks

A CNN [45] is a specialised ANN deep learning model which uses a convolutional operation to extract different patterns and features from input data. The standard CNN consists of a convolution layer, which has the objective of extracting high-level features and patterns from the input data, a pooling layer which reduces the dimension size of the features obtained by the convolutional layer, and a FCL which obtains the output of the model.

CNNs have become popular machine learning methods for text classification tasks after Kim [35] introduced the TextCNN approach which outperformed previous state-of-the-art approaches in various text classification tasks. Suppose we have a representation of a document as $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T] \in \mathbb{R}^{T \times d}$, where T is the number of tokens in the document and $\mathbf{x}_t \in \mathbb{R}^d \; \forall \, t \in \{1, \ldots, T\}$ is the d-dimensional word embedding for the t-th token in the sequence. The convolutional layer extracts patterns and features from this input text sequence with a filter $\mathbf{w} \in \mathbb{R}^{s \cdot d}$ where s determines the window size of the filter, i.e., how many consecutive tokens it groups together as it steps (or slides) over the input sequence. Figure 2.7 depicts this procedure of stepping over the input sequence for the sentence "The quick brown fox jumps over the lazy dog" with a window size of $s = 4$. The figure shows that during the first and second steps

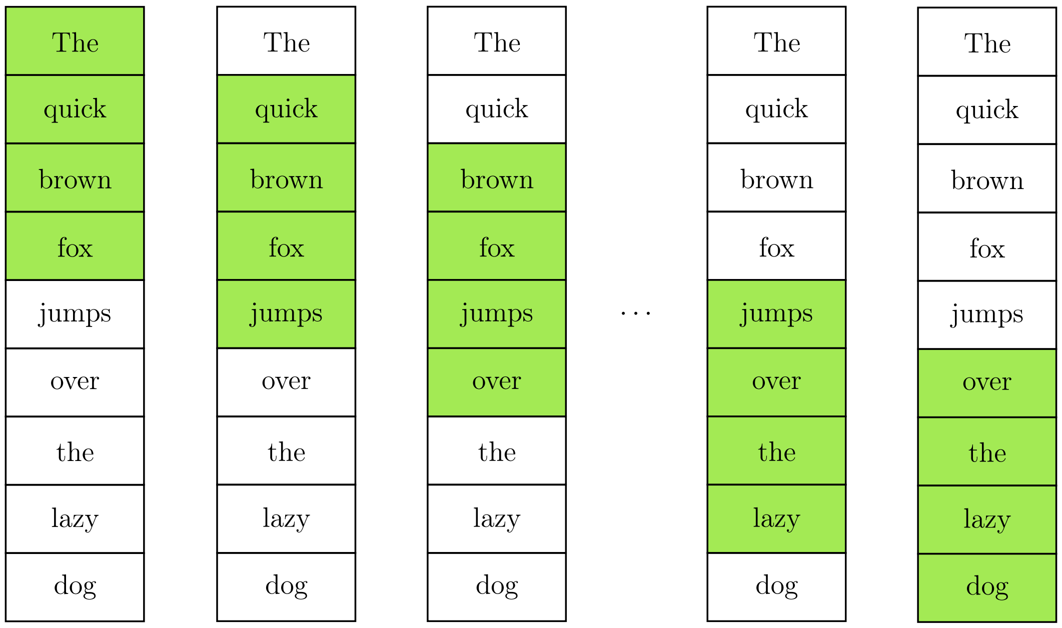


Figure 2.7: Example of the convolutional filter sliding over input sequence of tokens.

the filter groups together "The quick brown fox" and "quick brown fox jumps" respectively and continues this procedure until the final token is reached.

The filter $\mathbf{w}$ is randomly initialised with weights that are subsequently learnt during training. At each step, a feature value is obtained for the group of tokens under the filter by calculating the weighted sum of the embeddings where the weights are the values of the filter. Therefore, the feature value c_t for the t-th step is calculated as:

$$c_t = f(\mathbf{w} \cdot \mathbf{x}_{t:t+s-1} + b) \tag{2.5.1}$$

where f is an activation function, b is the bias unit, and $\mathbf{x}_{t:t+s-1}$ is the concatenated sequence of word embeddings $\mathbf{x}_t, \mathbf{x}_{t+1}, \ldots, \mathbf{x}_{t+s-1}$. After the filter has been applied to each position, a feature map is obtained as:

$$\mathbf{c} = [c_1, \ldots, c_{T-s+1}] \tag{2.5.2}$$

In practice, the CNN uses a certain number of these filters such that each filter learns different patterns from the input token sequence to produce a number of feature maps. Then, the pooling layer uses a summary statistic of the feature maps obtained by the convolutional layer to decrease the dimension of the representation and the required computational effort. Max-pooling is the most popular pooling function which simply obtains the maximum value of a particular feature. The output of the pooling layer is passed to the FCL which provides the output of the model. The CNN learns the weights of the different filters and the FCL to minimise an error function based on the output of the FCL and the expected output.

2.5.2 Recurrent Neural Networks

RNNs [29] are a class of ANNs which maintain an internal state or memory that captures information from previous timesteps in a sequence such that sequential data can be modelled. Suppose we have a representation of a document as $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$, where T is the number of tokens in the document and $\mathbf{x}_t \forall t \in \{1, \ldots, T\}$ is the word embedding for the t-th token in the sequence. At a certain timestep t, the model considers the input $(\mathbf{x}_t)$ as well as the hidden state of the previous layer $(\mathbf{h}_{t-1})$ to calculate the hidden state of the current timestep as:

$$\mathbf{h}_t = f(\mathbf{W}^{(h)}\mathbf{h}_{t-1} + \mathbf{W}^{(x)}\mathbf{x}_t) \tag{2.5.3}$$

where f is the activation function and $\mathbf{W}^{(h)}$ and $\mathbf{W}^{(x)}$ are weight matrices associated with the previous hidden state and the current input respectively. Thereafter, the hidden state $(\mathbf{h}_t)$ is passed through another FCL to obtain the output for timestep t. Equation (2.5.3) is used to determine the hidden state for each timestep such that each hidden state considers the hidden state of the previous timestep. Therefore, the final hidden state $(\mathbf{h}_T)$ in effect considers all of the previous hidden states of the sequence such that a document can be represented as a whole by the final hidden state. Typically, this final hidden state is passed through a FCL to obtain the output for text classification tasks.

The outputs for each timestep of the RNN are used along with their expected outputs in an error function to obtain the error (or loss) of the network. This error is used to learn the weights of the RNN through a generalisation of the backpropagation algorithm called backpropagation through time (BPTT) [91]. The BPTT algorithm considers the contributions from every timestep by backpropagating the gradient of the error function from the final step to the first step in the sequence.

The standard RNN architecture is prone to suffer from the vanishing gradient problem which is a result of the BPTT procedure that continuously multiplies derivatives back through the network such that the gradients become very small and effectively "vanish" over time [28]. This implies that the network struggles to consider the outputs of the earlier timesteps when sequences grow long. To combat the vanishing gradient problem, other RNN architectures such as the long short-term memory network (LSTM) [28] and gated recurrent unit network (GRU) [13] have been proposed which allow information to be preserved over longer sequences.

2.5.2.1 Long Short-Term Memory Networks

Along with the hidden state $(\mathbf{h}_t)$, LSTMs [28] maintain a memory state, referred to as the cell state, which has the goal of storing important long-term information. This information is maintained by three gate vectors: an input $(\mathbf{i}_t)$, forget $(\mathbf{f}_t)$, and output $(\mathbf{o}_t)$ gate. The input gate determines how much incoming information is used to update the cell state and the forget gate determines which information is retained from the previous cell state. The output gate determines which information from the cell state is used in the hidden state. These gates use a sigmoid activation function such that their associated vectors are calculated as:

$$\mathbf{i}_t = \sigma(\mathbf{W}^{(i)}\mathbf{h}_{t-1} + \mathbf{U}^{(i)}\mathbf{x}_t + \mathbf{b}^{(i)}) \tag{2.5.4}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}^{(f)}\mathbf{h}_{t-1} + \mathbf{U}^{(f)}\mathbf{x}_t + \mathbf{b}^{(f)}) \tag{2.5.5}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^{(o)}\mathbf{h}_{t-1} + \mathbf{U}^{(o)}\mathbf{x}_t + \mathbf{b}^{(o)}) \tag{2.5.6}$$

where the $\mathbf{W}$ and $\mathbf{U}$ matrices are weight matrices and the $\mathbf{b}$ vectors are bias vectors associated with each of the different gates. The new cell information ($\tilde{\mathbf{c}}_t$) combines the information from the previous hidden state ($\mathbf{h}_{t-1}$) and the input ($\mathbf{x}_t$) such that it is calculated as:

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}^{(c)}\mathbf{h}_{t-1} + \mathbf{U}^{(c)}\mathbf{x}_t + \mathbf{b}^{(c)}) \tag{2.5.7}$$

The cell state ($\mathbf{c}_t$) is calculated by using the forget gate to remove information from the previous cell state, and the input gate to add information from the new cell information as:

$$\mathbf{c}_t = \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \tilde{\mathbf{c}}_t \tag{2.5.8}$$

where $\otimes$ is the element-wise multiplication operator. Finally, the output gate is used to determine which parts of the cell state should be written to the hidden state as:

$$\mathbf{h}_t = \mathbf{o}_t \otimes \tanh(\mathbf{c}_t) \tag{2.5.9}$$

2.5.2.2 Gated Recurrent Unit Networks

The GRU [13] is another variant of RNNs which was proposed to prevent the vanishing gradient problem. GRUs use a similar strategy to LSTMs, but are simpler, have fewer parameters, and are more computationally efficient. GRUs do not maintain a cell state such as LSTMs and simply use two gates to control the information stored in the hidden state to retain information in long sequences. These gates are referred to as the reset gate ($\mathbf{r}_t$) and update gate ($\mathbf{u}_t$) which are calculated as:

$$\mathbf{r}_t = \sigma(\mathbf{W}^{(r)}\mathbf{h}_{t-1} + \mathbf{U}^{(r)}\mathbf{x}_t + \mathbf{b}^{(r)}) \tag{2.5.10}$$

$$\mathbf{u}_t = \sigma(\mathbf{W}^{(u)}\mathbf{h}_{t-1} + \mathbf{U}^{(u)}\mathbf{x}_t + \mathbf{b}^{(u)}) \tag{2.5.11}$$

where the $\mathbf{W}$ and $\mathbf{U}$ matrices are weight matrices and the $\mathbf{b}$ vectors are bias vectors associated with the different gates. The reset gate determines which parts of the previous hidden state should be retained when the new candidate hidden state ($\tilde{\mathbf{h}}_t$) is calculated as:

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}^{(h)}(\mathbf{r}_t \otimes \mathbf{h}_{t-1}) + \mathbf{U}^{(h)}\mathbf{x}_t + \mathbf{b}^{(h)}) \tag{2.5.12}$$

The update gate controls the amount of information that should be used from the previous hidden state ($\mathbf{h}_{t-1}$) and the candidate hidden state ($\tilde{\mathbf{h}}_t$) to calculate the current hidden state as:

$$\mathbf{h}_t = (1 - \mathbf{u}_t) \otimes \mathbf{h}_{t-1} + \mathbf{u}_t \otimes \tilde{\mathbf{h}}_t \qquad (2.5.13)$$

2.5.3 Attention Mechanisms

The attention mechanism was first proposed by Bahdanau *et al.* [2] for the task of machine translation, i.e., language translation through machine learning algorithms. To give context on their approach we first describe the standard encoder-decoder RNN-based architecture for machine translations tasks. Furthermore, we describe the general definition of attention and multi-head attention as proposed by Vaswani *et al.* [84].

2.5.3.1 Encoder-Decoder Architectures

The standard RNN encoder-decoder or sequence-to-sequence (seq2seq) architecture consists of two RNNs known as the encoder and decoder respectively. The encoder RNN receives the sequence of tokens as input and obtains a context vector as the final hidden state which provides a representation of all the information in the input sequence. The decoder RNN initialises its first hidden state with the context vector obtained by the encoder and its first input token as a special token which indicates the start of a sequence. This special token is passed to the decoder RNN which obtains the hidden state as a combination of the previous hidden state and the input. The hidden state is passed through a FCL with a softmax activation function over the target language vocabulary. Therefore, the FCL calculates a probability distribution over each word in the target language and typically selects the word with the highest probability as the input to the following timestep in the decoder RNN.

The process of passing the predicted output as the input to the following timestep is continued until a special end-of-sequence token is predicted by the decoder RNN. Therefore, each timestep uses the hidden states and output predictions of the previous timesteps to predict the next word and eventually form the translation. However, this architecture struggles to effectively model longer input sequences since the decoder RNN only uses the final hidden state of the encoder RNN such that it fails to capture all of the information contained in longer input sequences [2].

2.5.3.2 Attention in Encoder-Decoder Architectures

The attention mechanism proposed by Bahdanau *et al.* [2] modifies the standard RNN seq2seq architecture to address the decrease in performance for longer input sequences. The intuition behind the attention mechanism for machine translation is based on the way humans translate text, by placing attention on specific words in the text input while considering the overall context of the input. Suppose the hidden states of the encoder RNN were obtained as $\mathbf{H}^{(e)} = [\mathbf{h}_1{}^{(e)}, \ldots, \mathbf{h}_T{}^{(e)}]$, where T is the number of tokens in the input sequence. The standard RNN seq2seq approach uses the final hidden state ($\mathbf{h}_T{}^{(e)}$) as the context vector which is used to initialise the hidden state of the decoder RNN.

Bahdanau *et al.* [2] proposed a method which obtains a context vector for each timestep as a weighted sum over all of the encoder hidden states. At a timestep t, the weights associated with each encoder hidden state are calculated by obtaining an alignment score between the previous decoder hidden state ($\mathbf{h}_{t-1}{}^{(d)}$) and each of the encoder hidden states, and passing the result through a softmax function such that the weights sum to one. The weights are referred to as attention weights and are calculated as:

$$\boldsymbol{\alpha} = \text{softmax}\left(a(\mathbf{h}_{t-1}{}^{(d)}, \mathbf{H}^{(e)})\right) \tag{2.5.14}$$

where a is an alignment function which measures the similarity between the previous decoder hidden state and all of the encoder hidden states to determine which encoder hidden states should receive higher attention weights. The softmax function for the i-th element of an input $\mathbf{t}$ is defined as:

$$\text{softmax}(\mathbf{t})_i = \frac{e^{t_i}}{\sum_{j=1}^{D} e^{t_j}} \tag{2.5.15}$$

where D is the number of elements in $\mathbf{t}$.

The attention weights are used to calculate the context vector for timestep t as:

$$\mathbf{c}_t = \boldsymbol{\alpha}\mathbf{H}^{(e)} \tag{2.5.16}$$

where $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_T]$ such that $\alpha_j \forall j \in \{1, \ldots, T\}$ represents the amount of attention placed on encoder hidden state $\mathbf{h}_j{}^{(e)}$ in the current decoder timestep. Finally, the current decoder hidden state ($\mathbf{h}_t^{(d)}$) is calculated as a function of the previous hidden state ($\mathbf{h}_{t-1}^{(d)}$), the context vector ($\mathbf{c}_t$), and the input at the current timestep, i.e., the output of the previous timestep.

2.5.3.3 General Definition of Attention

Even though the attention mechanism was originally developed for encoder-decoder RNN architectures, a general definition of these mechanisms is provided by Vaswani *et al.* [84]. In the generalised definition of an attention mechanism we consider a query $\mathbf{Q}$ and a set of key-value pairs $\mathbf{K}$ and $\mathbf{V}$. Attention weights ($\boldsymbol{\alpha}$) are computed between the query $\mathbf{Q}$ and each of the keys in $\mathbf{K}$ by calculating an alignment score and passing the result through a softmax function,

$$\boldsymbol{\alpha} = \text{softmax}\left(a(\mathbf{Q}, \mathbf{K})\right) \tag{2.5.17}$$

where a is called an alignment (or compatibility) function which aims to measure the similarity between the given query and key to determine which parts of the vector $\mathbf{V}$ related to that key to place attention on. The two alignment functions used in this thesis are referred to as the dot product alignment function given by:

$$a(\mathbf{Q}, \mathbf{K}) = \mathbf{Q}\mathbf{K}^T \tag{2.5.18}$$

and the general alignment function given by:

$$a(\mathbf{Q}, \mathbf{K}) = \mathbf{Q}\mathbf{W}\mathbf{K}^T \tag{2.5.19}$$

where $\mathbf{W}$ is learnt during training.

The output is calculated as a weighted sum over the values in $\mathbf{V}$ and $\boldsymbol{\alpha}$ given by:

$$\mathbf{c} = \boldsymbol{\alpha}\mathbf{V} \tag{2.5.20}$$

The intuition behind this process is that each element of $\boldsymbol{\alpha}$ represents how much attention is placed on each element of $\mathbf{V}$.

2.5.3.4 Multi-head Attention

Vaswani *et al.* [84] also proposed multi-head attention, which is an extension of the attention mechanism that applies multiple attention functions over a set of queries, keys, and values. The multi-head attention mechanism uses several attention heads whose inputs are obtained by projecting the original queries, keys, and values. For the i-th attention head, the input query, key, and value matrices are calculated as:

$$\mathbf{Q}_i = \mathbf{Q}\mathbf{W}_i^{(Q)} \tag{2.5.21}$$

$$\mathbf{K}_i = \mathbf{K}\mathbf{W}_i^{(K)} \tag{2.5.22}$$

$$\mathbf{V}_i = \mathbf{V}\mathbf{W}_i^{(V)} \tag{2.5.23}$$

where the $\mathbf{W}$ weights are learnt during training. Thereafter, an attention mechanism is performed for each head with independent parameters. The final output of the multi-head attention mechanism is obtained by concatenating the context vectors from each of the attention heads. By using multiple attention heads, the network is able to attend to different parts of the input sequence through several attention patterns. Therefore, the different heads can focus on different characteristics of the input and thereby extract more complex and useful information [84].

2.5.4 Transformers

Vaswani *et al.* [84] originally proposed the transformer ANN architecture for machine translation tasks, and obtained state-of-the art performance on several benchmark machine translation datasets. However, this architecture has

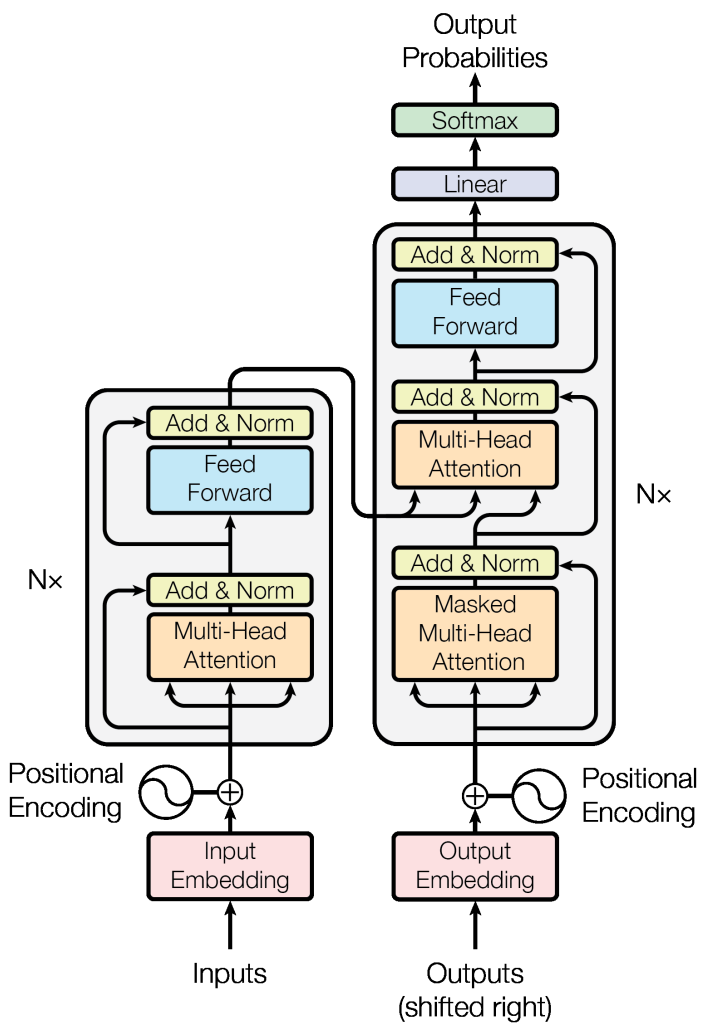


Figure 2.8: Transformer model architecture, taken from Vaswani *et al.* [84]. Encoder (left): Input tokens are converted through an embedding layer and combined with positional embeddings. These embeddings are passed through a multi-head attention mechanism with a residual connection followed by a FFNN to obtain the final encoder representation of each token. Decoder (right): The decoder uses a masked multi-head attention mechanism followed by the encoder-decoder attention mechanism which uses the final encoder token representations as queries. Finally, the output is passed through a FFNN and another linear layer with a softmax function to determine the following token in the output sequence which forms the input at the following timestep.

led to a paradigm shift in the field of NLP, with transformer-based language models such as BERT [17], GPT [66], and others significantly outperforming previous state-of-the-art approaches in various tasks. Transformers use a seq2seq architecture as depicted in Figure 2.8, where an encoder obtains a representation of the input sequence and passes this representation to the decoder which obtains the output of the model. Furthermore, the transformer architecture does not use convolutions or recurrence but rather leverages the effectiveness of the attention mechanism in the encoder and decoder modules. In this thesis we only use transformer-based model architectures that comprise transformer encoder modules, but we provide a high-level explanation of the decoder module in this section for sake of completeness.

2.5.4.1 Encoder

The left part of Figure 2.8 depicts the encoder module which is used to encode representations of the input token sequence. Suppose we have a sequence of tokens given as $\mathbf{x} = [x_1, \ldots, x_T]$, where T is the number of tokens in the sequence. These tokens are passed through an embedding layer that maps each token to a word embedding such that the sequence is given by $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$. The embedding layer can be represented as a matrix, where each row contains the word embedding for a particular token. Therefore, each token in the vocabulary, i.e., the set of unique tokens, is assigned a unique token ID which corresponds to the row of its word embedding in the embedding matrix.

Due to the model effectively ignoring the sequential structure of the input, positional vectors $(\mathbf{p}_1, \ldots, \mathbf{p}_T)$ are added to the word embeddings in order to explicitly model the positions of the words in the sequence. The positional vectors can be set explicitly or be learnt during training and are represented as a matrix in a similar manner to the embedding layer such that position IDs are mapped to the rows of the matrix which contains the positional embeddings for the different token positions. Therefore, the input token x_t is converted to an embedding which is calculated as:

$$\mathbf{e}_t = \mathbf{x}_t + \mathbf{p}_t \tag{2.5.24}$$

such that the input token sequence is converted to a matrix $\mathbf{E} = [\mathbf{e}_1, \ldots, \mathbf{e}_T]$.

Each encoder module comprises two sub-layers, namely the multi-head self-attention mechanism and a position-wise FFNN. Figure 2.9 depicts the high-level architecture of the multi-head self-attention layer. In the multi-head self-attention mechanism the query, key, and value matrices are all chosen as the input matrix such that $\mathbf{Q} = \mathbf{K} = \mathbf{V} = \mathbf{E}$. Therefore, the i-th attention head ($head_i$) is calculated as:

$$\mathbf{Q}_i = \mathbf{E}\mathbf{W}_i^{(Q)} \tag{2.5.25}$$

$$\mathbf{K}_i = \mathbf{E}\mathbf{W}_i^{(K)} \tag{2.5.26}$$

$$\mathbf{V}_i = \mathbf{E}\mathbf{W}_i^{(V)} \tag{2.5.27}$$

$$head_i = \text{softmax}\left(\frac{\mathbf{Q}_i{\mathbf{K}_i}^T}{\sqrt{d_k}}\right)\mathbf{V}_i \tag{2.5.28}$$

where d_k is the dimension of the key vectors and the scaling factor $\frac{1}{\sqrt{d_k}}$ is used to prevent the dot product between the queries and keys from becoming too

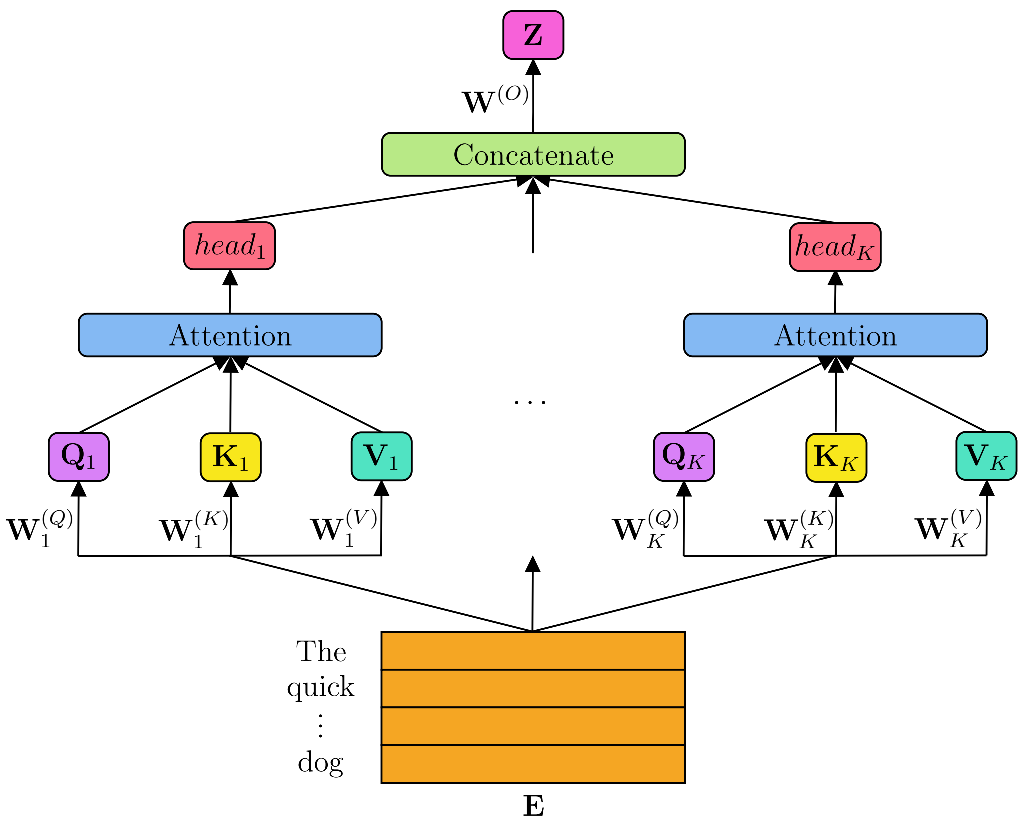


Figure 2.9: Multi-head self-attention mechanism in transformer encoder module. The input embeddings ($\mathbf{E}$) are passed to K attention heads where they are projected to queries ($\mathbf{Q}$), keys ($\mathbf{K}$), and values ($\mathbf{V}$) through the $\mathbf{W}^{(Q)}$, $\mathbf{W}^{(K)}$, and $\mathbf{W}^{(V)}$ weight matrices respectively. The queries, keys, and values are passed through an attention mechanism to obtain the output of the specific head. Finally, the output of all the heads are concatenated and projected with the $\mathbf{W}^{(O)}$ matrix to obtain the output of the multi-head self-attention mechanism as the matrix $\mathbf{Z}$.

large since this would force the softmax function into regions with very small gradients [2]. The output of these attention heads are combined to encode the input sequence as:

$$\mathbf{Z} = Concat(head_1, \dots, head_K)\mathbf{W}^{(O)} \tag{2.5.29}$$

where $Concat$ concatenates the vectors from each of the K attention heads and $\mathbf{W}^{(O)}$ is learnt during training. Thereafter, the input matrix $\mathbf{E}$ is added to the output of the multi-head self-attention layer which is known as a residual connection [25] and has shown to improve the performance of transformer models [84]. Therefore, the output of the multi-head self-attention layer is calculated as:

$$\mathbf{A} = \mathbf{E} + \mathbf{Z} \tag{2.5.30}$$

where the rows of the matrix $\mathbf{A}$ are given as $\mathbf{a}_1, \dots, \mathbf{a}_T$ and represent the multi-head self-attention outputs for each token.

The output of the multi-head self-attention layer is passed through a position-wise FFNN with a residual connection to obtain the hidden states for each token x_t as:

$$\mathbf{h}_t = \mathbf{a}_t + ReLU(\mathbf{a}_t\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2 \tag{2.5.31}$$

where the weights ($\mathbf{W}$) and biases ($\mathbf{b}$) are randomly initialised and learnt during training. The hidden representations of all the tokens $(\mathbf{h}_1, \ldots, \mathbf{h}_T)$ are passed to the following encoder module and so forth until the final encoder representation for each token is obtained. Furthermore, layer normalisation [1] is applied to the output of each of these sub-layers such that the following layer is less sensitive to the scale of inputs which improves the stability and convergence speed during training.

2.5.4.2 Decoder

As shown on the right side of Figure 2.8, the decoder module has a similar structure to the encoder module, but uses an additional attention layer, referred to as the encoder-decoder attention layer, that is driven by the final representation of the encoder hidden states. The input to the decoder at the first timestep is a special start-of-sequence token. However, the following timesteps use a token predicted by the previous timestep as its input. These input tokens are passed through an embedding layer and combined with positional encodings as in the encoder. Thereafter, the embeddings are passed to a masked multi-head self-attention mechanism which is similar to the multi-head self-attention mechanism in the encoder module, but uses a masking matrix which allows the attention mechanism to only attend to tokens it has previously predicted in the output sequence.

The output of the masked multi-head self-attention layer is used as the query matrix in the multi-head encoder-decoder attention mechanism where the encoder representations are used as the key and value matrices. Therefore, the encoder-decoder attention layer obtains the weighted sum of the encoder hidden states which are conditioned based on the output of the masked multi-head self-attention layer. The output of the encoder-decoder attention layer is passed through a position-wise FFNN which obtains the decoder hidden state. Finally, the hidden state is passed to a FCL with a softmax activation function which provides a probability distribution over the vocabulary of tokens. The probability distribution is used to select a token from the vocabulary to form the input in the following timestep.

2.6 Transformer-based Language Models

In recent years the field of NLP has undergone a significant paradigm shift due to the advent of language models which are pre-trained through self-supervised learning tasks on unlabelled textual data. Self-supervised learning is a machine learning paradigm which aims to learn from unlabelled data by automatically creating objectives or labels from the unlabelled data. For example, the model can hide a certain section of the input data and attempt to predict the missing section such that it learns the structure and patterns of the unlabelled data during training.

Language models are typically pre-trained through self-supervised tasks on a large amount of textual data with the objective of obtaining a general understanding of language by creating representations of text tokens which capture their semantic and contextual information. The pre-trained models are usually fine-tuned on downstream tasks which use labelled data to adjust the parameters of the pre-trained model for the specific task. The rationale behind fine-tuning a pre-trained model is that the pre-trained model obtains natural language understanding capabilities during the pre-training phase which serves as a good starting point for downstream NLP tasks. Therefore, the knowledge obtained during the pre-training phase can be transferred and leveraged during the fine-tuning phase to perform downstream tasks more effectively. Models that follow this paradigm of fine-tuning PLMs, have outperformed previous state-of-the-art approaches in various NLP tasks [14, 17, 101].

Many language models have been proposed in recent years, but we discuss those that are most relevant to this thesis which include: BERT, RoBERTa, ELECTRA, and DeBERTaV3.

2.6.1 BERT

Devlin *et al.* [17] proposed the Bidirectional Encoder Representations from Transformers (BERT) architecture which comprises a number of transformer encoder modules stacked on top of each other. The BERT architecture transforms the input sequence of tokens to word embeddings with added positional encodings as in the transformer architecture. BERT learns representations for each token which captures its bidirectional context and semantics through two self-supervised tasks, namely masked language modelling (MLM) and next sentence prediction (NSP).

Figure 2.10 depicts the high-level procedure of the MLM task used during BERT pre-training. The model uses the MLM pre-training task to mask a certain percentage of the input tokens with the objective of predicting their true tokens during training in order to obtain a general language understanding. Therefore, a certain percentage of tokens are replaced with a special `[MASK]` token after which the sequence is passed to the model which obtains the final hidden state for each token as the output of the last encoder module. These final hidden states capture the bidirectional context and semantics of each token through the stacked transformer encoder modules. The final hidden states of the `[MASK]` tokens are passed through a FCL with a softmax function to obtain the output as a distribution over the entire vocabulary of tokens. The

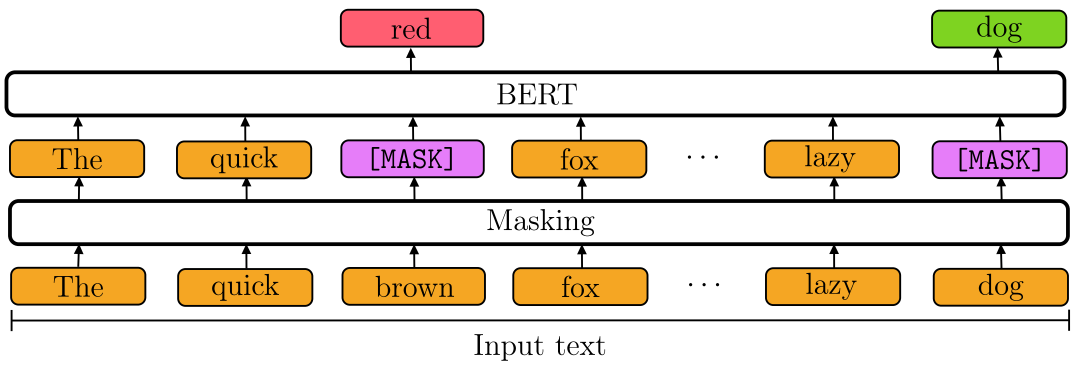


Figure 2.10: MLM task. Input text tokens (orange) are randomly replaced with `[MASK]` tokens (purple) and passed to the BERT model which predicts the masked tokens correctly (green) or incorrectly (red) to drive the training process.

output of each `[MASK]` token is compared to its true token through an error function which drives the training process.

Devlin *et al.* choose 15% of the input tokens as candidates to be masked, but only mask 80% of the candidate tokens with the `[MASK]` token. 10% of the candidate tokens are replaced with a random token and the last 10% of the candidate tokens remain unchanged. This is done in order to prevent the mismatch between the pre-training and fine-tuning phases, since downstream tasks generally do not use the MLM training objective. This forces the model to maintain a contextual representation of each token in the sequence, because it does not know whether a token has been masked, randomly replaced, or is unchanged. Since only 1.5% of tokens are randomly replaced, the ability of the model to understand language is not hindered [17]. Therefore, the MLM pre-training task allows the model to form language understanding capabilities through the creation of embeddings which represent each token in the input sequence.

The BERT architecture also uses the NSP task to train the model to understand relationships between texts by switching sequences of input tokens 50% of the time and instructing the model to predict which sequence should come first. They differentiate between the two sequences of contiguous tokens by separating them with a special `[SEP]` token and adding segment embeddings which represent whether they are in the first or second span of tokens. Furthermore, each sequence which is passed to the model is prepended with a special classification token `[CLS]` whose final hidden state is used to predict whether the two text sequences follow each other or not.

Through comprehensive experiments Devlin *et al.* showed that their model outperformed previous state-of-the-art approaches on various NLP tasks after fine-tuning BERT for those tasks.

2.6.2 RoBERTa

Zhuang *et al.* [101] proposed the Robustly optimised BERT pre-training approach (RoBERTa) which improved the original BERT model through various techniques. They proposed several model architecture adaptions such as removing the NSP pre-training task and applying dynamic masking to the training data. The original BERT implementation randomly masked tokens during the pre-processing procedure such that the masks stayed the same over the epochs of training and therefore had a static masking strategy. The dynamic masking strategy randomly masks tokens of each sequence before it is passed to the model and was shown to be important for performance when pre-training for longer and with more data [101].

Zhuang *et al.* performed comprehensive experiments to compare various aspects of the pre-training phase and further improved on the original BERT model through using larger batch sizes, a larger pre-training data volume, and increased training time. Their approach obtained state-of-the-art results on several NLP benchmark tasks and illustrated the significance of the design decisions of the BERT architecture.

2.6.3 ELECTRA

Clark *et al.* [14] proposed a transformer-based language model called Efficiently Learning an Encoder that Classifies Token Replacements Accurately (ELECTRA). ELECTRA uses a generator-discriminator framework, where a generator corrupts the input token sequence, while the discriminator has the objective of distinguishing between original tokens and tokens that have been corrupted by the generator. The discriminator and generator both have a transformer-based encoder architecture which converts a sequence of input tokens to a sequence of vector representations that capture the contextual information of each token. Figure 2.11 depicts the high-level architecture of ELECTRA during pre-training.

The generator is trained to perform the MLM task as in the BERT architecture, by randomly masking a percentage of tokens and learning to predict the masked tokens. Therefore, the original input sequence is passed to the generator, which converts certain tokens to `[MASK]` tokens and "fills" them with predicted tokens to create a corrupted token sequence.

The corrupted token sequence is passed to the discriminator which uses the replaced token detection (RTD) task to predict which tokens were replaced by classifying each token as `original` or `replaced`. This is done by passing the final hidden state of each token through a FCL which outputs a score that represents the probability of the token being `original` or `replaced`. Finally, the outputs are compared to the true outputs in an error function which drives the training of the discriminator.

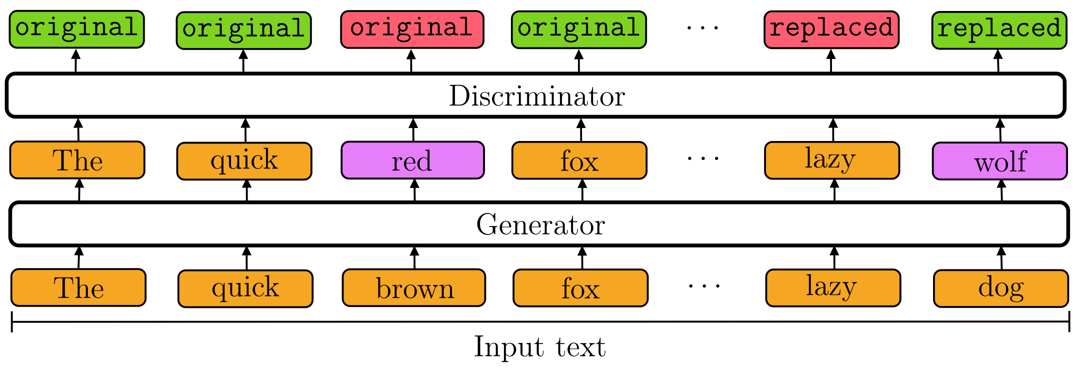


Figure 2.11: ELECTRA architecture during pre-training. The input text tokens (orange) are passed to the generator which corrupts certain tokens (purple) and passes the corrupted token sequence to the discriminator. The discriminator predicts each token as being `original` or `replaced` and uses the correct (green) and incorrect (red) predictions to drive the training process.

During pre-training the parameters of the embedding layer are shared between the generator and discriminator to leverage the effectiveness of the MLM task in learning word embeddings. After the pre-training phase, the generator is typically removed and the discriminator is used as a PLM which can be fine-tuned on downstream tasks. Clark *et al.* show that the RTD pre-training task is much more sample efficient than MLM approaches since it forces the model to consider each word in the input sequence.

2.6.4 DeBERTaV3

He *et al.* [27] proposed the Decoding-enhanced BERT with disentangled attention (DeBERTa) approach which improves the BERT architecture through two novel techniques. The DeBERTa model uses a disentangled attention mechanism, where each token is represented with two embeddings that capture its content and relative positional information respectively. Therefore, the attention weights between tokens are calculated using separate (or disentangled) matrices on their content and positional representations. The reasoning behind this approach is based on the observation that attention weights of tokens often depend strongly on relative positions, and not only the contents of the tokens. The relative positional encodings differ from the standard absolute positions which assign a unique positional embedding for each token to capture the absolute position of the tokens in the sequence. Alternatively, the relative positional encodings are dynamic and context-dependant embeddings that capture the relative positions of the tokens in a sequence by considering the distance or relative position of the tokens in a certain window. Furthermore, DeBERTa uses an enhanced mask decoder which incorporates the absolute positions of tokens in the decoding layer during the MLM pre-training task [27]. He *et al.* showed that their proposed improvements significantly improve the effectiveness of the pre-training phase as well as performance on downstream tasks.

Later, He *et al.* [26] proposed DeBERTaV3 which improves on DeBERTa by replacing the MLM pre-training task with RTD. However, they observed that the embedding sharing techniques proposed in ELECTRA hinders the performance of the model since the training errors of the discriminator and generator "pull token embeddings in different directions" [26]. Therefore, DeBERTaV3 uses a gradient-disentangled embedding sharing method between the generator and discriminator which improves the efficiency of training and model performance. The gradient-disentangled embedding sharing method shares the embedding layer parameters between the generator and discriminator as in the ELECTRA model but does not allow the error of the discriminator to influence the optimisation of the generator, thereby preventing the "tug-of-war" dynamics which are caused by the conflicting objectives of the generator and discriminator. He *et al.* show that DeBERTaV3 obtains state-of-the-art performance on various NLP tasks and generally outperforms BERT, RoBERTa, and ELECTRA.

2.6.5 Model Sizes

The transformer-based PLMs described above are typically created in three different model sizes which are `small`, `base`, and `large`. These different model sizes allow users to find an effective trade-off between performance and computational requirements, since larger models generally perform better but are computationally more expensive [17]. These model sizes differ in the number of layers (transformer encoder modules), the size of the hidden states and the number of attention heads. In this thesis we only use `base` models which have 12 layers, a hidden state size of 768, and 12 attention heads which results in a total of 86 million model parameters.

2.6.6 Sequence Length Limits

The multi-head self-attention mechanism in the transformer-based PLM requires the computation of an attention score between each token and every other token in the input sequence. This quadratic scaling of computations leads to these models becoming computationally expensive as the sequence length grows long. Therefore, sequence length limits are imposed such that only a certain number of tokens are allowed to form part of the input sequence. The sequence lengths limits are controlled by the size of the positional encoding matrix which only has a certain number of rows mapping position IDs to position embeddings. Therefore, sequences that are longer than this predefined size are truncated. All of the PLMs used in this thesis use a sequence length limit of 512 tokens.

2.7 Graph Neural Networks

Graph neural networks (GNNs) are machine learning algorithms that have the objective of representing the nodes of a graph structure with vectors (or embeddings) which capture the properties of the nodes and the connections between them [87]. GNNs have been used to obtain vector representations of hierarchical class structures after which these representations are incorporated into the classification procedure through various techniques [10, 32, 89, 90, 100]. Many GNN architectures have been proposed, but we describe the most relevant architectures that have been used in previous HTC approaches: the Graph Convolutional Network (GCN) [37], the Graph Attention Network (GAT) [85], and the Graphormer [98]. These approaches all use the idea of "message passing", where the nodes of the graph send and receive information from neighbouring nodes. However, these architectures differ in how the graph nodes aggregate the information from their neighbouring nodes in order to update their associated node embeddings.

Suppose we have a graph structure with N nodes and each node has an associated feature vector that represents the information and characteristics of that node. The features can be represented as a matrix $\mathbf{F}$, where each row $\mathbf{f}_i \forall\, i \in \{1, \ldots, N\}$ corresponds to the i-th node's feature vector.

A GCN [37] consists of a number of layers which have the objective of aggregating information between the nodes in the graph such that the final node embeddings capture the graph structure information and relationships between the nodes. The GCN updates the feature vector for node i at a specific layer (l) as:

$$\mathbf{f}_i^{(l+1)} = f\left(\sum_{j \in \mathcal{N}(i)} \frac{\mathbf{f}_j^{(l)} \mathbf{W}^{(l)}}{\sqrt{\deg(i)\deg(j)}}\right) \tag{2.7.1}$$

where f is a chosen activation function, $\mathcal{N}(i)$ returns the set of neighbours of node i, $\mathbf{W}^{(l)}$ is a set of learnable weights for layer l, and $\deg(i)$ is the degree of node i, i.e., the number of nodes connected to node i (similarly for node j). These representations are passed through several layers with the same structure after which the final layer obtains the representations of each node in the graph.

A GAT [85] uses a different approach to aggregate the information from neighbouring nodes. At a particular layer l, the GAT calculates an attention score for each node i to all of its neighbouring nodes $j \in \mathcal{N}(i)$ and to itself as:

$$e_{i,j}^{(l)} = f\left((\mathbf{f}_i^{(l)} \mathbf{W}^{(l)})^T \cdot (\mathbf{f}_j^{(l)} \mathbf{W}^{(l)})\right) \tag{2.7.2}$$

These scores are passed through a softmax function such that the attention scores for a particular node to its neighbouring nodes and itself sums to 1. The converted attention weights are given as $\alpha_{i,j}^{(l)}$. The vector representation for each node is updated by aggregating the features from neighbouring nodes with their respective attention weights as:

$$\mathbf{f}_i^{(l+1)} = f\left(\sum_{j\in\mathcal{N}(i)\cup\{i\}} \alpha_{i,j}^{(l)} \cdot \mathbf{f}_j^{(l)}\mathbf{W}^{(l)}\right) \tag{2.7.3}$$

The Graphormer [98] architecture uses several approaches for incorporating the graph structural information into the transformer architecture. Firstly, the Graphormer adds centrality encodings to each node representation such that it captures the indegree and outdegree for node i as:

$$\mathbf{f}_i^{(0)} = \mathbf{f}_i^{(0)} + \mathbf{z}^-_{\text{deg}^-(i)} + \mathbf{z}^+_{\text{deg}^+(i)} \tag{2.7.4}$$

where $\mathbf{z}^-_{\text{deg}^-(i)}$ and $\mathbf{z}^+_{\text{deg}^+(i)}$ are learnable vectors which represent the information associated with the indegree and outdegree of node i respectively. These added centrality encodings enable the attention mechanisms to capture the importance of the node in the graph through its indegree and outdegree.

The Graphormer architecture also adds a spatial encoding to encode the structural information of the graph into the model. The spatial encoding uses a function $\phi(i,j)$ which measures the spatial distance between connected nodes i and j in the graph as the distance of the shortest path between the nodes. For nodes that are not connected $\phi(i,j) = -1$. For each feasible output value of $\phi(i,j)$, the model assigns a learnable scalar value $b_{\phi(i,j)}$ which is used as the bias term in the attention weight calculation. Therefore, each $\phi(i,j)$ has a corresponding bias term $b_{\phi(i,j)}$ which represents the spatial distance between the two nodes such that the attention scores are calculated as:

$$e_{i,j}^{(l)} = \frac{(\mathbf{f}_i^{(l)}\mathbf{W}^{(Q)})(\mathbf{f}_j^{(l)}\mathbf{W}^{(K)})^T}{\sqrt{d}} + b_{\phi(i,j)} \tag{2.7.5}$$

where d is the dimension of the vectors. Therefore, the Graphormer allows each node to attend to each of the other nodes in the graph according to the graph's structural information that is modelled by $b_{\phi(i,j)}$. The attention scores are passed through a softmax function to obtain the attention weights which are used to update the node vectors through the standard transformer architecture with multi-head self-attention and FFNN layers.

2.8 Topic Models

Topic models [6] are unsupervised machine learning models that have the objective of extracting abstract topics from a corpus of documents. The extracted topics are typically represented as a distribution over the vocabulary found in the corpus such that this distribution indicates the most important words to distinguish between the different topics. These topics can be used to represent a document or a particular word in a document as a distribution over the set of topics such that the distribution represents the most relevant topics associated with the document or word. Several topic modelling approaches have been proposed which include older approaches such as Latent Dirichlet Allocation (LDA) [7] and more recent approaches which use PLMs such as BERTopic [21].

LDA [7] is a topic model based on the BOW approach which represents a text document as a multi-set of its words and ignores ordinal and contextual information. The LDA model represents each topic as a probabilistic distribution over a set of words contained in the corpus of documents. The documents are represented as a probabilistic distribution over a set of topics and therefore the most relevant topics associated with each document can be determined based on the assumption that documents with similar topics will use a similar group of words. The LDA algorithm is provided with a fixed number of K topics to discover in a corpus of documents and has the objective of learning the topic distribution of each document and the word distribution of each topic. Initially, the LDA algorithm randomly assigns each word in a document to a topic. These assignments of words to topics are used to form the topic representations as the distribution of the words assigned to each topic. Similarly, the document representations are formed as the distribution of the topics assigned to the words of the document. The algorithm attempts to reassign topics to words such that the topic representations become representative of the topics found in the corpus of documents. This is done through the following procedure:

1. For every document $d_i \forall\, i \in \{1, \ldots, N\}$, where N is the number of documents in the corpus, calculate the proportion of words in the document that are currently assigned to each topic $T \in \{1, \ldots, K\}$ as $p(\text{Topic } T \mid \text{Document } d_i)$.

2. For every word $w \in \{1, \ldots, V\}$, where V is the size of the corpus vocabulary, calculate the proportion of assignments to the topic T as $p(\text{Word } w \mid \text{Topic } T)$.

3. For each word w in document d_i, compute a confidence score for each topic as $p(\text{Topic } T \mid \text{Document } d_i) \cdot p(\text{Word } w \mid \text{Topic } T)$ and assign the topic T with the highest confidence to that word.

4. Repeat the steps above until an approximately steady state of assignments is reached.

BERTopic [21] is a more recent topic extraction model which utilises PLMs and clustering techniques to extract abstract topics from a corpus of documents. Firstly, BERTopic uses a Sentence-BERT [68] model to convert each document into a fixed-sized vector representation which captures the semantics of the document. A sentence-BERT model is a modified BERT model that is tuned to produce a semantic embedding for a sentence or document as opposed to the standard BERT model that produces embeddings for each token in the input sequence. This is done by passing two text sequences separately through the same BERT model and applying a contrastive loss function [22] such that semantically similar sequences are pulled closer while semantically dissimilar sequences are forced further apart in the output embedding space. The BERTopic model reduces the dimensionality of these document representations with the UMAP [57] dimensionality reduction algorithm and clusters the resulting representations with the HDBSCAN [56] algorithm, such that each cluster of documents represents a topic. Finally, BERTopic uses these clusters to find the topic-word distributions by concatenating all of the documents in a cluster and applying the TF-IDF procedure as described in Section 2.4.3 to find the most relevant words for the associated clusters. BERTopic uses the TF-IDF scores of each word to represent the distribution of words for the topic.

2.9 Evaluation Metrics

Evaluation metrics are used to measure the performance of machine learning models and are crucial to understanding which models perform better for certain tasks. Unsuitable evaluation metrics may lead to incorrect conclusions in terms of model performance. To evaluate a classification model, we use a test set of instances with their associated true class assignments. The test instances are passed to the model which obtains the predicted outputs after which the performance of the model is evaluated through an evaluation metric which compares the predicted output of the model to the true output.

Due to the hierarchical structure of classes in HTC tasks, the choice of evaluation metrics is not trivial. Several evaluation metrics have been proposed for hierarchical classification tasks, but they are often only used in the paper in which it was proposed [78]. However, the most common evaluation metrics used for hierarchical classification tasks are the Micro-F1 and Macro-F1 scores [10, 16, 31, 32, 89, 90, 100]. Therefore, we use these evaluation metrics in this thesis to compare different classification approaches. Note that for these evaluation metrics we “flatten” the class hierarchy and do not consider the hierarchical class structure during the evaluation phase. We describe the general procedure for calculating the Micro-F1 score and then generalise this to the Macro-F1 score.

Consider a true positive (TP) and a false positive (FP) as a correctly and incorrectly assigned class respectively. Similarly, consider a true negative (TN) and a false negative (FN) as a correctly and incorrectly unassigned category respectively. We represent the total number of TP, FP, TN and FN assignments from the set of test instances as TP_{total}, FP_{total}, TN_{total} and FN_{total} respectively.

Precision (P) is defined as the proportion of correctly assigned classes to the total number of assigned classes, i.e., $\frac{\text{TP}_{\text{total}}}{\text{TP}_{\text{total}}+\text{FP}_{\text{total}}}$. Therefore, it measures how accurate an assigned class is on average and is calculated as:

$$P = \frac{1}{N}\sum_{i=1}^{N}\frac{|\mathbf{y}_i \cap \hat{\mathbf{y}}_i|}{|\hat{\mathbf{y}}_i|} \tag{2.9.1}$$

where N is the number of test instances and $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$ denote the true and predicted classes for document i respectively.

Recall (R) is defined as the proportion of correctly predicted classes to the total number of true classes, i.e., $\frac{\text{TP}_{\text{total}}}{\text{TP}_{\text{total}}+\text{FN}_{\text{total}}}$. Therefore, it measures how many of the true class assignments the model captures on average and is calculated as:

$$R = \frac{1}{N}\sum_{i=1}^{N}\frac{|\mathbf{y}_i \cap \hat{\mathbf{y}}_i|}{|\mathbf{y}_i|} \tag{2.9.2}$$

Micro-F1 combines P and R by taking their harmonic mean to provide a comprehensive measure of the performance which balances the precision and recall. Therefore, higher Micro-F1 values indicate better performance, and is calculated as:

$$\text{Micro-F1} = \frac{2 \times P \times R}{P + R} \tag{2.9.3}$$

Micro-F1 simply averages the performance of the model over all of the test instances, and is therefore less suitable for cases where there is a class imbalance among the test instances, i.e., some classes have significantly more instances than others. Therefore, Macro-F1 is used which separately obtains a P, R, and F1 score for each class and takes their arithmetic mean such that equal weight is given to each of the classes. Suppose we have the total number of TP, FP, TN, and FN assignments for class $j \in \{1, \ldots, L\}$, where L is the number of classes, as TP_j, FP_j, TN_j, and FN_j respectively. The Macro-F1 score is calculated as:

$$P_j = \frac{\text{TP}_j}{\text{TP}_j + \text{FP}_j} \tag{2.9.4}$$

$$R_j = \frac{\text{TP}_j}{\text{TP}_j + \text{FN}_j} \tag{2.9.5}$$

$$\text{Macro-F1} = \frac{1}{L}\sum_{j=1}^{L}\frac{2 \times P_j \times R_j}{P_j + R_j} \tag{2.9.6}$$

2.10 Benchmark Datasets

Several datasets have been consistently used to evaluate the performance of HTC approaches. The three most commonly used datasets which are most relevant to this thesis include: Web Of Science (WOS) [41], Reuters Corpus Volume 1 Version 2 (RCV1-V2) [46] and New York Times (NYT) [73]. Tables 2.1 and 2.2 provide summary statistics and hierarchical properties of these datasets. The dataset splits (Train, Development, and Test) were proposed by Zhou *et al.* [100] who sampled the splits from the original data. These splits are typically used in recently proposed HTC approaches [31, 32, 89, 90, 100]. For comparison reasons we therefore use the same dataset splits in this thesis. In this section, we describe the procedures followed to create these three datasets and provide further details on each of them. Furthermore, we briefly describe the International Classification of Diseases (ICD) and MIMIC datasets since many flat classifier approaches have been proposed for ICD code assignment tasks. However, we do not consider the ICD code classification tasks in this thesis since they are specific to the medical domain and often comprise very long document lengths which make them impractical or infeasible for standard PLM-based classification approaches. Therefore, we focus on the standard HTC tasks with shorter document lengths in broader domains such as news articles and research publications.

Table 2.1: Characteristics of the three benchmark HTC datasets. The columns "Levels" and "Classes" give the number of levels and classes in the class structure. "Avg. Classes" is the average number of classes per document, while "Train", "Dev", and "Test" are the number of instances in each of the dataset splits.

Dataset	Levels	Classes	Avg. Classes	Train	Dev	Test
WOS	2	141	2.0	30,070	7,518	9,397
RCV1-V2	4	103	3.24	20,833	2,316	781,265
NYT	8	166	7.6	23,345	5,834	7,292

Table 2.2: Average per-level branching factor of the hierarchy in each of the three benchmark datasets, which is calculated as the average number of child nodes for the nodes at a particular level. The number of nodes per level is given in parentheses.

Dataset	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6	Level 7	Level 8
WOS	19.14 (7)	0.0 (134)	–	–	–	–	–	–
RCV1-V2	13.75 (4)	0.78 (55)	0.02 (43)	0.0 (1)	–	–	–	–
NYT	6.75 (4)	1.89 (27)	0.92 (51)	0.36 (47)	0.71 (17)	0.5 (12)	0.33 (6)	0.0 (2)

2.10.1 Web of Science

The WOS dataset was constructed by Kowsari *et al.* [41] who collected 46,985 research publication abstracts from the Web of Science publication database. A detailed methodology of how they created the dataset is not available, but they collected the abstract, subject domain, and keywords for each publication. They created a two-level hierarchical class structure by using the subject domains as classes for the first level of the hierarchy and the keywords as the second level classes.

Figure 2.12 presents the first-level classes of the WOS dataset with the associated number of children classes and number of documents assigned to each class, while Figure 2.13 presents the distribution of the number of documents assigned to each of the second-level classes. Figure 2.13 shows that most second-level classes have between 300 and 400 documents. However, on the extreme edge of the class distribution, the "Electrical Generator" and "Structured Storage" classes only have 1 and 43 document assignments respectively. On the other side, the classes "Molecular Biology" and "Polymerase Chain Reaction" have 746 and 750 documents respectively.

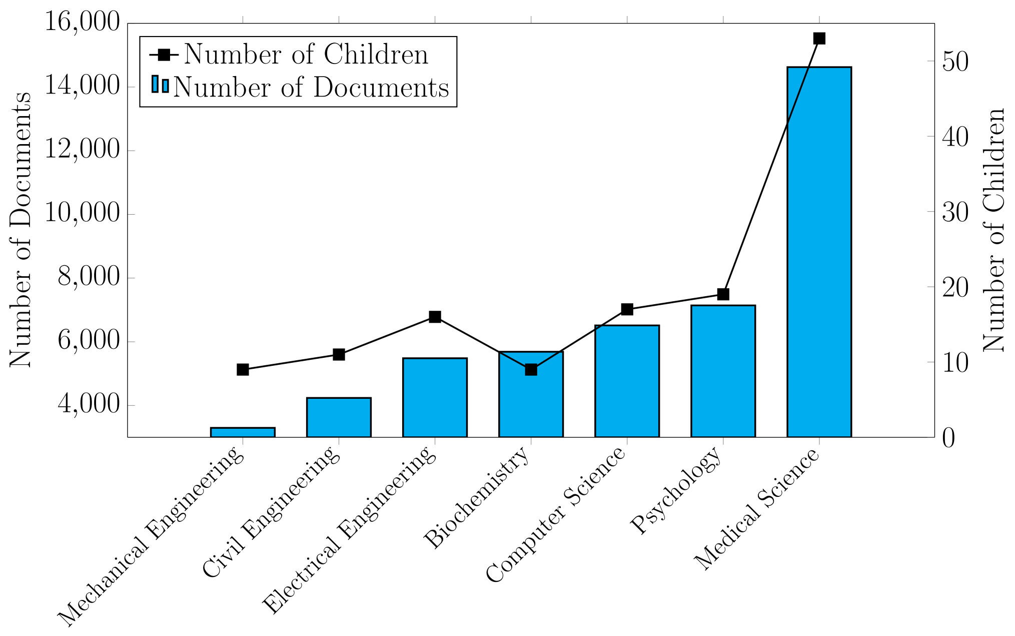


Figure 2.12: Characteristics of first-level classes of the WOS dataset. The bar plot gives the number of documents assigned to each class (left y-axis) while the line plot shows the number of children for each class (right y-axis).

2.10.2 RCV1-V2

The Reuters Corpus Volume 1 Version 1 (RCV1-V1) [70] is a dataset that consists of 806,791 English news articles published by Reuters, Ltd. between August 1996 and August 1997. Each article was categorised into classes from a four-level hierarchical class structure by a rule-based text categorisation system, known as the Topic Identification System (TIS) [24]. Furthermore, categories that were previously manually assigned from journalists were mapped to

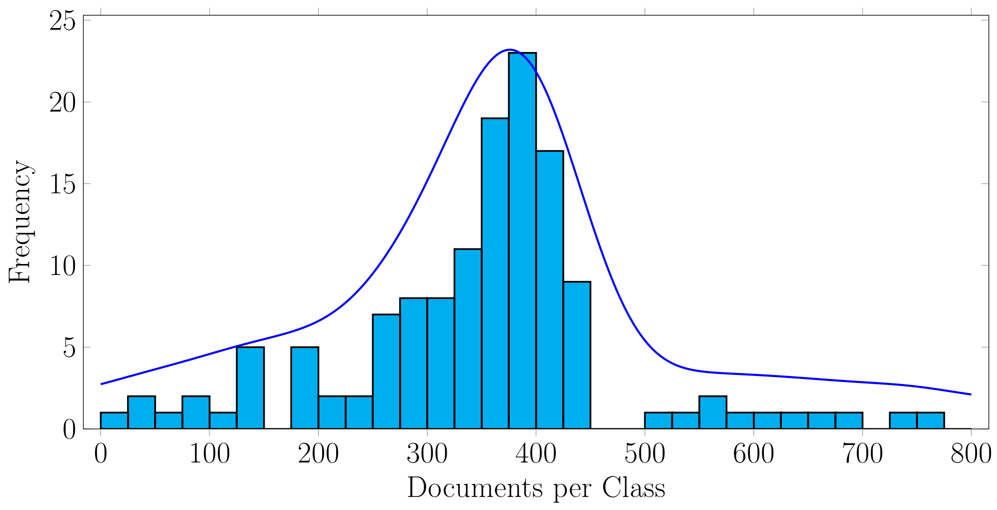


Figure 2.13: Distribution of the number of documents assigned to each of the second-level WOS classes.

the same classification schema and combined with the TIS classifications [46]. Thereafter, each article was manually inspected by at least one editor who added classifications to articles with no classes and corrected mistakes in the codings. Lewis *et al.* [46] state that 79% of articles had at least one of the automatic classifications overruled, showing that the dataset is closer to a manually categorised dataset than an algorithmically assigned one.

Lewis *et al.* [46] created the RCV1-V2 dataset which addressed several issues of the RCV1-V1 dataset by removing articles that do not have any categories assigned to them and adding all missing ancestor classifications from the hierarchical class structure. The RCV1-V2 dataset contains 804,414 instances with 103 classes spread over four levels.

Figure 2.14 presents the first-level classes of the RCV1-V2 dataset with the associated number of children classes and number of documents assigned to each class, while Figure 2.15 presents the distribution of the number of documents assigned to each of the lower-level classes. From these figures we can see that the document assignments are extremely unbalanced. For example, the non-first-level classes with the fewest document instances are “Millennium Issues” and “European Community (General)” with 5 and 40 documents respectively. Similarly, on the other extreme edge, the classes “Commodity Markets” and “Performance” have 85,440 and 151,783 documents respectively.

2.10.3 NYT

The New York Times Annotated Corpus [73] is a dataset which contains 1,855,658 news articles published by the New York Times between January 1987 and June 2007. Each article is accompanied with metadata such as the date of publication, keywords, article summary, and classifications. The clas-

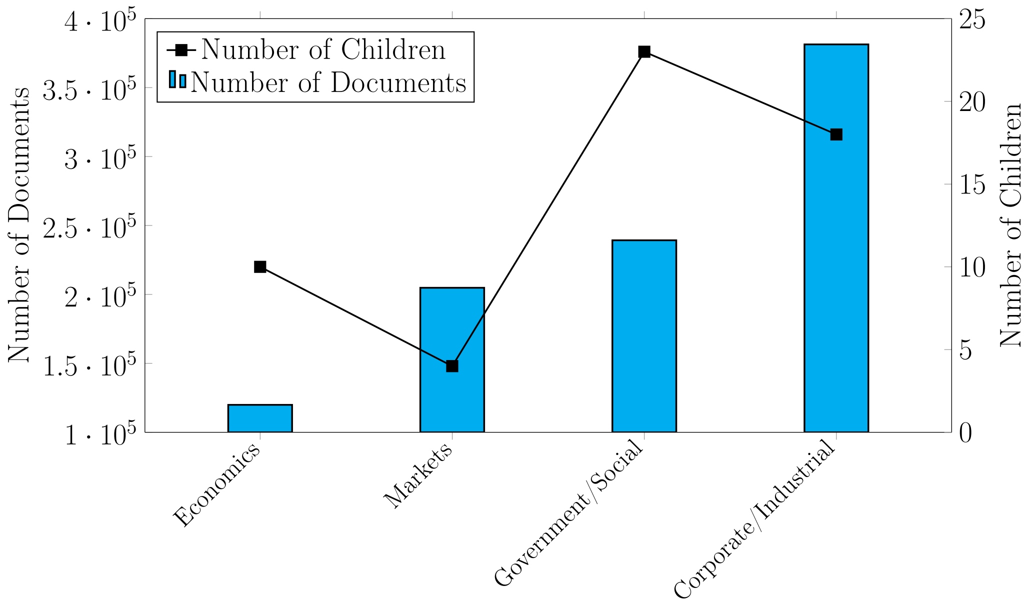


Figure 2.14: Characteristics of first-level classes of the RCV1-V2 dataset. The bar plot gives the number of documents assigned to each class (left y-axis) while the line plot shows the number of children for each class (right y-axis).

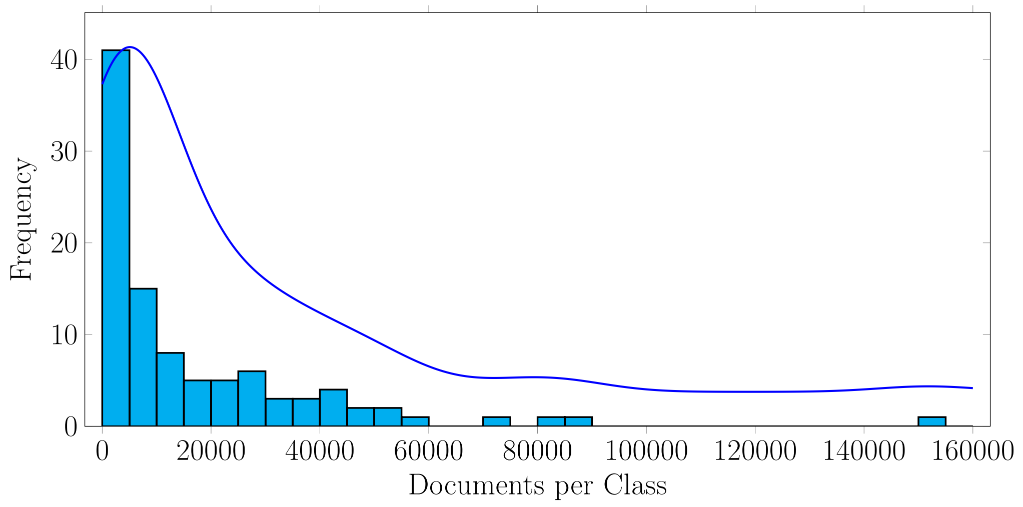


Figure 2.15: Distribution of the number of documents assigned to each of the non-first-level RCV1-V2 classes.

sification schema of the articles forms a structured class hierarchy with 8 levels and 166 classes. The classifications were algorithmically assigned to each article and manually verified by the New York Times staff. However, the algorithm used to assign the classification has not been published. The NYT dataset was created by randomly sampling 36,471 article summaries from the New York Times Annotated Corpus along with their assigned classes [100].

Figure 2.16 presents the first-level classes of the NYT dataset with the associated number of children classes and number of documents assigned to each class, while Figure 2.17 presents the distribution of the number of documents assigned to each of the lower-level classes. Figure 2.17 shows the imbalance at the lower-levels of the class hierarchy with classes such as “Circuits” and “Iran” only comprising 143 and 157 documents respectively while the “Guides” and “Travel” classes have 15,142 and 15,296 documents respectively.

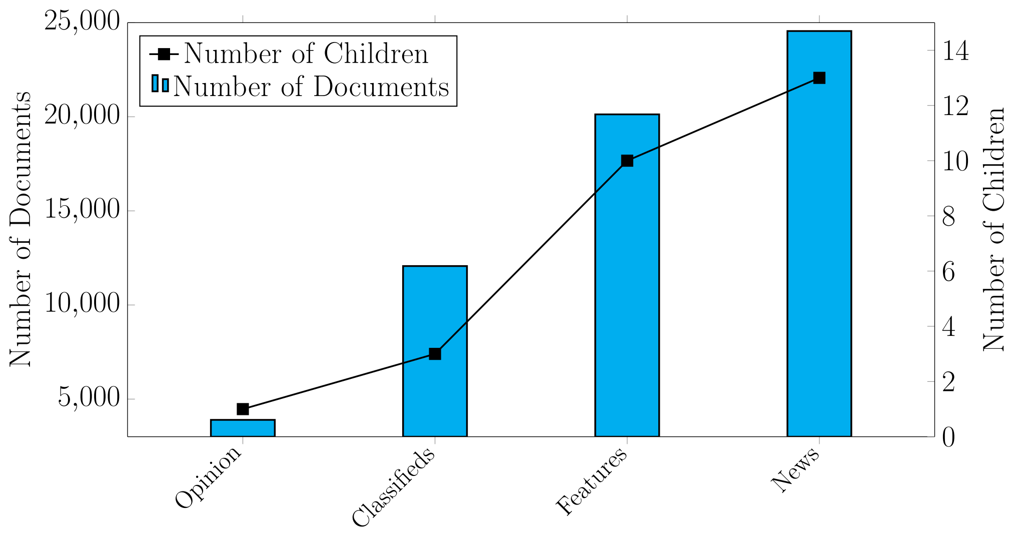


Figure 2.16: Characteristics of first-level classes of the NYT dataset. The bar plot gives the number of documents assigned to each class (left y-axis) while the line plot shows the number of children for each class (right y-axis).

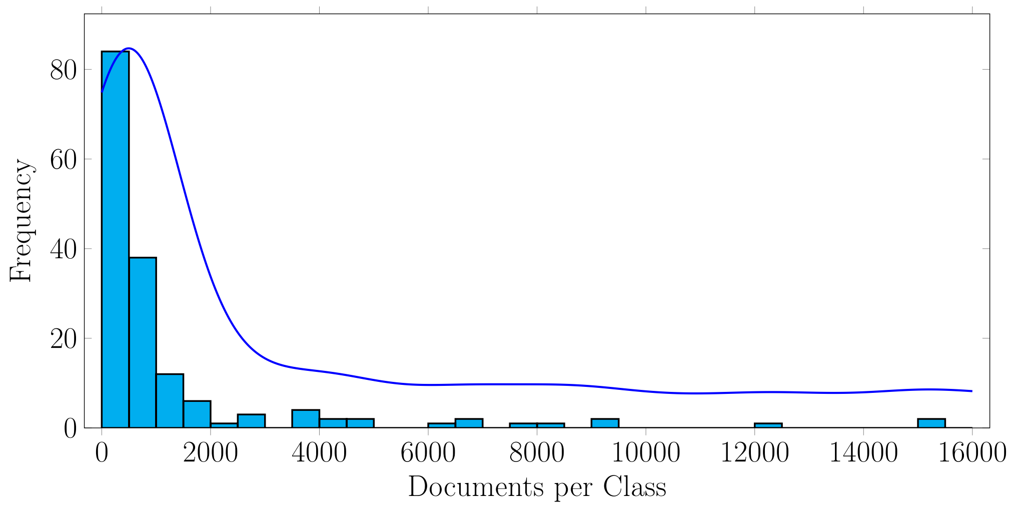


Figure 2.17: Distribution of the number of documents assigned to each of the non-first-level NYT classes.

2.10.4 ICD and MIMIC

The International Classification of Diseases (ICD) is a diagnosis coding set maintained by the World Health Organisation. The ICD system provides a hierarchical structure of diagnosis codes which describes the diagnosis of a patient. The MIMIC-II and MIMIC-III datasets contain medical records of patients that were admitted to critical care units at the Beth Israel Deaconess Medical Center [34]. These records contain detailed information such as laboratory tests, diagnosis codes, and clinical text notes written by medical professionals. A common HTC research topic is to use the clinical text notes to automatically assign ICD codes to a patient record. The MIMIC-II dataset contains 22,815 patient records with associated clinical text notes whereas the updated MIMIC-III dataset contains 52,722 patients records with associated clinical text notes. Furthermore, MIMIC-III-50 is a dataset which only uses the 50 most common ICD codes in the MIMIC-III dataset. The median number of words for the documents in the MIMIC-II and MIMIC-III datasets are 1,322 and 1,375 respectively. Since the sequence length limit of transformer-based PLMs is typically only 512 tokens, we do not use these datasets in this thesis.

Chapter 3

Literature Review

This chapter presents the related research on HTC approaches that are most relevant to this thesis. As described in Section 2.3, the three main strategies which are used to solve HTC problems include: flat, local, and global classifier approaches.

3.1 Flat Classifier Approaches

Flat classifier approaches ignore the hierarchical class structure and transform the HTC task into a multi-label text classification (MLTC) task. One specific HTC application for which the flat classifier approach has been commonly adopted is the categorisation of medical text notes from the MIMIC datasets into ICD codes [4, 47, 50, 59, 75, 80, 86]. Furthermore, many approaches have been proposed for solving standard MLTC tasks but are also applied to HTC datasets such as RCV1-V2. Therefore, any MLTC approach can also be seen as a flat classifier approach. In this section, we first describe several flat classifier approaches that were developed for the ICD code assignment tasks and thereafter describe the most relevant approaches that were developed for MLTC tasks but have been applied to HTC datasets such as RCV1-V2.

3.1.1 ICD Code Assignment Approaches

The flat classifier approaches proposed for ICD code assignment tasks use various deep learning architectures such as CNNs [47, 59], RNNs [4, 75, 86], and transformer-based PLMs [50, 80]. Table 3.1 presents the results of the approaches discussed in this section on the three most commonly used datasets for the medical text note classification task which include: MIMIC-II, MIMIC-III, and MIMIC-III-50.

In 2017, Shi *et al.* [75] proposed a method for medical text note classification which uses the textual descriptions of the class set along with attention mechanisms to solve the MLTC task. They used pre-processing techniques to

Table 3.1: Summary performance results of flat classifier approaches on the three most commonly used datasets for assigning ICD codes to medical text notes.

Model	Year	MIMIC-II		MIMIC-III		MIMIC-III-50	
		Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
C-LSTM-ATT [75]	2017	–	–	–	–	53.20	–
HA-GRU [4]	2017	36.60	–	40.50	–	–	–
CAML [59]	2018	44.20	4.80	53.90	8.80	61.40	53.20
DR-CAML [59]	2018	45.70	4.90	52.90	8.60	63.30	57.60
MultiResCNN [47]	2019	46.40	5.20	55.20	8.50	67.00	60.60
LAAT [86]	2020	48.60	5.90	57.50	9.90	71.50	66.60
JointLAAT [86]	2020	49.10	**6.80**	57.50	**10.70**	71.60	66.10
HiLAT [50]	2022	–	–	–	–	**73.50**	**69.00**
M-XLNet [80]	2023	**49.89**	–	**59.38**	–	71.70	–

extract several diagnoses from a single patient record and passed each diagnosis through character and word-level LSTM models to encode the diagnosis as a vector representation. More specifically, the character-level LSTM encodes the characters of the text sequence into word representations. Thereafter, the word representations are encoded to document representations through the word-level LSTM. Similarly, the textual descriptions of the classes are encoded with LSTMs into a vector representation. These class representations capture the semantics of each ICD code by encoding their textual descriptions through the LSTMs. Finally, an attention mechanism is used to match the vector representations of the text document and the ICD codes to determine the most suitable assignments for the medical text note. Their approach obtained a Micro-F1 score of 53.20 on the MIMIC-III-50 dataset.

Baumel *et al.* [4] proposed the Gated Recurrent Unit with a Hierarchical Attention mechanism (HA-GRU) model in 2017. The HA-GRU model breaks the text document up into sentences and uses a hierarchical set of GRUs to obtain document representations. First, HA-GRU uses a GRU to encode the words of each sentence, and uses an attention mechanism to obtain sentence embeddings for each sentence as a weighted average of the word embeddings in the sentence. Therefore, each sentence embedding captures the most important words of that sentence by placing more weight on their associated word embeddings. Similarly, the sentence embeddings are passed through another GRU and attention mechanism to obtain the document representation as the weighted average of the sentence embeddings. However, the attention mechanism in the second level GRU learns a separate query vector for each class to obtain different document representations which capture the most important features for each class. This is known as a label-wise attention mechanism since it allows the model to derive label-specific representations of the document where each class assigns more weight to the features of the document that are important for classifying that particular class. These label-specific document representations are passed through separate FCLs associated with each of the classes. Finally, the results from the classification layers are combined to determine the classes which are assigned to a text instance. The HA-GRU model outperformed previous-state-of-the art approaches with Micro-F1 scores of 36.30 and 40.50 on the MIMIC-II and MIMIC-III datasets.

In 2018, Mullenbach *et al.* [59] improved on the HA-GRU approach by proposing the Convolutional Attention for Multi-Label classification (CAML) model. They used the word2vec CBOW method to obtain word embeddings for each word in a document which are used as input to a CNN classifier. CAML uses a standard CNN architecture for text classification as introduced by Kim [35], but replaces the max-pooling layer with a label-wise attention mechanism. The attention mechanism obtains label-specific weighted averages of the features derived from the convolutional layer by determining the most important features for each of the classes individually. These label-wise document representations are passed through associated FCLs and used to perform binary classification for each class. Furthermore, they proposed the Description Regularised CAML (DR-CAML) model which uses a second CNN model to learn representations for the textual descriptions of the classes. These vector representations are used to guide a regularisation function which drives the parameters of rare classes to be more similar to more frequent classes with similar class descriptions. Therefore, DR-CAML aims to improve performance on rare classes by leveraging the parameters of more frequent classes that have similar textual descriptions and therefore should be semantically similar. The DR-CAML approach achieved state-of-the-art performance in terms of Micro-F1 and Macro-F1 on the MIMIC-II (45.70 and 4.90) and MIMIC-III-50 (63.30 and 57.60) datasets but was outperformed by CAML on MIMIC-III.

In 2019, Li and Yu [47] improved the CAML approach with two techniques to create the Multi-Filter Residual Convolutional Neural Network (MultiResCNN) approach. Firstly, the model uses a convolutional layer with multiple channels where each channel has a different filter size such that they group together a different number of words as they step over the input sequence. This allows the convolutional layer to extract different granularities of the text document's patterns and features to improve classification performance. Furthermore, MultiResCNN applies the residual mechanism [25] by adding a second convolutional layer to each of the convolutional layer channels. They showed that the MultiResCNN model outperforms previous approaches such as CAML and DR-CAML on the MIMIC datasets with Micro-F1 scores of 46.40, 55.20, and 67.00 and Macro-F1 scores of 5.20, 8.50, and 60.60 on the MIMIC-II, MIMIC-III, and MIMIC-III-50 datasets respectively.

Vu *et al.* [86] developed the Label Attention Model (LAAT) in 2020. LAAT uses an LSTM to encode the tokens in a document and passes the token representations to a label-wise attention mechanism to obtain label-specific representations of the document. These label-specific representations are used in separate classification layers as in the approaches described above. However, LAAT uses an adaptation of the general attention function as opposed to the dot product attention used in CAML and MultiResCNN. The LAAT approach outperformed previous-state-of-the-art approaches such as MultiResCNN over the three MIMIC datasets by obtaining Micro-F1 scores of 48.60, 57.50, and 71.50 and Macro-F1 scores of 5.90, 9.90, and 66.60 on the MIMIC-II, MIMIC-III, and MIMIC-III-50 datasets respectively.

In the same paper, Vu *et al.* [86] also proposed the JointLAAT approach which uses a separate attention mechanism for the different levels of the class hierarchy and projects the predictions of the classes at a certain level to a vector representation. These prediction representations are concatenated to the feature vectors used in the lower level attention mechanism in order to leverage the parent-level predictions to improve classification performance. JointLAAT generally outperformed the LAAT approach on the three MIMIC datasets with Micro-F1 scores of 49.10, 57.50, and 71.60 and Macro-F1 scores of 6.80, 10.70, and 66.10 on the MIMIC-II, MIMIC-III, and MIMIC-III-50 datasets respectively. These results indicate that the use of higher-level prediction representations improves classification performance. Note that JointLAAT is not a flat classifier approach since it does not ignore the class hierarchy, but we include it here for sake of clarity and comparison.

In 2022, Liu *et al.* [50] proposed the Hierarchical Label-wise Attention Transformer (HiLAT) model which splits the text document into chunks and uses token-and chunk-level label-wise attention mechanisms sequentially. They use the XLNet language model [95] which is pre-trained through the permutation language modelling task by randomly permuting the order of tokens in the input sequence and training the model to predict tokens using both the original and permuted orders as context. This approach allows the model to capture the bidirectional context of the token sequence while retaining the ability of modelling the joint probability of the sequence since there are no masked tokens in the input sequence. Furthermore, by removing the need for `[MASK]` tokens during pre-training, XLNet avoids the discrepancy between the pre-training and fine-tuning phases since there are typically no masked tokens in the fine-tuning phase.

HiLAT passes the chunks of the documents separately through the XLNet model to obtain semantic and contextual representations of each token in each chunk. In the token-level label-wise attention mechanism, the token representations of each chunk are transformed into label-specific representations of the associated chunk. The label-wise representations from each chunk are combined and passed to the chunk-level attention mechanism which obtains a document representation for each class. HiLAT uses the same label-wise attention mechanism as proposed by Vu *et al.* [86] for the LAAT model. Finally, these label-wise document representations are passed through associated FCLs to obtain the confidence score for each class. HiLAT achieved state-of-the-art performance on the MIMIC-III-50 dataset in terms of Micro-F1 (73.50) and Macro-F1 (69.00).

Most recently, Strydom *et al.* [80] proposed the Multilabel XLNet (M-XLNet) approach which adds the label-wise attention mechanism from CAML to a pre-trained XLNet language model [95] and fine-tunes the model on the downstream dataset. First, they pre-trained a small XLNet model on the MIMIC-III dataset to account for the domain-specific nature and the long sequence lengths of the medical text notes. During fine-tuning, the M-XLNet model obtains representations of each token in the input sequence and passes these representations to a label-wise attention mechanism to obtain label-specific representations of the document. Finally, these representations are passed through their associated FCLs to obtain the output scores for each of the classes which are combined to form the output of the model. Their approach achieved state-of-the-art Micro-F1 scores on the MIMIC-II (49.89) and MIMIC-III (59.38) datasets. However, HiLAT [50] outperformed M-XLNet on the MIMIC-III-50 Micro-F1 score and the results of HiLAT are not reported on the MIMIC-II and MIMIC-III datasets.

3.1.2 Multi-label Classification Approaches

Many approaches have been proposed for MLTC tasks in recent years which use various deep learning architectures such as CNNs [43, 49], RNNs [51, 60, 94], and transformer-based language models [52, 97]. However, recent approaches are often proposed for MLTC tasks which have extremely large class sets and are therefore not applicable to this thesis. Therefore, in this section we only describe a few recently proposed MLTC approaches which are the most relevant to this thesis. Unfortunately, these approaches are difficult to compare due to the use of different evaluation metrics, datasets, and dataset splits.

In 2019, Xiao *et al.* [94] proposed the Label Specific Attention Network (LSAN) for MLTC tasks. LSAN converts the document tokens into GloVe word embeddings and passes the embeddings through LSTMs to obtain a representation of the document. LSAN uses two label-wise attention functions to extract label-specific representations of the text documents. First, LSAN uses the label-wise attention mechanism proposed by Vu *et al.* [86] for the LAAT model to obtain label-specific representations which capture the most important features of the document content. The second label-wise attention mechanism leverages the textual descriptions of the class names by using their GloVe word embeddings in an attention function with the document encodings to obtain label-specific document representations which capture the semantic correlation between the document content and the class descriptions. Finally, these label-specific representations are combined and passed through FCLs to obtain the output of the model. LSAN outperformed five previous MLTC approaches on four benchmark datasets which included RCV1-V2.

Pal *et al.* [60] proposed the Multi-label Text classification using Attention based Graph Neural NETwork (MAGNET) approach in 2020. The MAGNET approach converts the document tokens to embeddings through the BERT embedding layer and passes the embeddings through LSTM models to obtain token representations which capture the context of the token sequence. Furthermore, MAGNET uses a GAT to model the correlation between the different classes. This is done by learning an adjacency matrix which represents the strength (or weight) of the connections between the different classes. Therefore, the GAT obtains representations of each class which capture the correlations with other classes. Finally, the correlation-aware class representations are multiplied with the document representation to obtain the predictions for each of the possible classes. Pal *et al.* showed that their approach outperformed previous state-of-the-art approaches on five benchmark MLTC datasets.

In 2021, Liu *et al.* [52] proposed an approach which combines the features extracted from a PLM and a topic model to train a CNN classifier for MLTC tasks. The rationale for combining the features obtained by these models is that the PLM extracts fine-grained semantic and contextual representations while the representations obtained by the topic model capture higher-level "global" information which takes the entire corpus of documents into account. The PLM is used to extract semantic word embeddings for each token in the document and the document as a whole. Furthermore, the topic model extracts vector representations for each word and document which represents the distribution over the set of abstract topics created by the topic model. They combine the extracted features from these models to train a CNN classifier which comprises multiple convolutional layers with different filter sizes to group varying lengths of token sequences together. Finally, their approach applies max-pooling to the output of the convolutional layers followed by a FCL to obtain the confidence scores of a document belonging to each of the possible classes. Liu *et al.* showed that their approach outperformed other MLTC approaches such as XML-CNN [49] and LSAN [94] on two benchmark datasets.

3.2 Local Classifier Approaches

Local classifier approaches attempt to leverage the hierarchical structure of the class taxonomy by creating multiple classifiers and combining their results in various ways to improve the classification performance on HTC tasks. The three main approaches, as described in Section 2.3, include the LCN, LCPN, and LCL approaches. Unfortunately, these approaches are difficult to compare directly due to inconsistent use of datasets and performance measures throughout the literature. However, the results in Table 3.2 give an overview of the results achieved by recent approaches on the most commonly used benchmark datasets.

In 1997, Koller and Sahami [39] proposed the first method for HTC which leverages the hierarchical structure of classes. They used the one-hot-encoding feature extraction method to obtain feature representations of the documents

Table 3.2: Summary performance results of local classifier approaches on the most commonly used datasets.

Model	Year	RCV1-V2		WOS
		Micro-F1	Macro-F1	Accuracy
HDLTex [41]	2017	–	–	76.58
HFT-CNN [76]	2018	80.29	51.40	63.71
HTrans [3]	2019	**80.51**	**58.49**	–
HE-HMTC [53]	2022	–	–	**78.51**

and a LCPN approach which constructs a classifier for each non-leaf node in the class hierarchy. Each of these classifiers are trained to distinguish between its children nodes and are only trained on the subset of training instances which belong to their associated class. During inference, instances start at the root node classifier which classifies them into a particular child node. The instances are sent to the classifier of their assigned node to be classified further and this procedure is continued until a leaf node is reached.

More recently (2017), Kowsari *et al.* [41] proposed Hierarchical Deep Learning for Text Classification (HDLTex) which uses the LCPN approach to build a classifier for each non-leaf node in the class hierarchy. They experimented with different types of deep learning classifiers such as RNNs and CNNs, as well as feature extraction methods such as TF-IDF scores and GloVe word embeddings. Similarly to the approach proposed by Koller and Sahami [39], HDLTex only uses the subset of training data where the parent node is positive to train the classifier at a particular node. Furthermore, Kowsari *et al.* constructed the WOS dataset and outperformed several previous approaches with the proposed HDLTex approach. However, the HDLTex approach may become computationally expensive for HTC tasks with a large number of internal nodes since it requires a deep learning classifier for each non-leaf node in the class hierarchy.

Stein *et al.* [79] performed several experiments to compare the performance of different word embedding generation methods such as word2vec and GloVe to obtain the semantic representations of the tokens in a document for HTC problems. Furthermore, they compared different classifier architectures such as SVM [33], CNN, fastText [9], and XGBoost [11]. Lastly, they compared the performance of the LCPN approach, which builds a classifier for each non-leaf node, to the flat classification approach which ignores the class hierarchy. They showed that the use of word embeddings significantly improved classification performance for HTC tasks and that the local classifier approach outperformed the flat classifier approach by leveraging the hierarchical structure of the class taxonomy.

Shimura *et al.* [76] proposed the Hierarchical Fine-Tuning based CNN (HFT-CNN) model in 2018. HFT-CNN uses a LCL approach to train a CNN classifier with max-pooling for each level of the class hierarchy. Initially, the first-level CNN classifier is trained to distinguish between the first-level classes of the class hierarchy. The model initialises the embedding layer and convolutional layer parameters of the second-level CNN classifier with the parameters of the first-level classifier and repeats this process until the final level of the hierarchy is reached. The rationale behind this design decision is that the parameters of higher levels serve as a good starting point for lower levels which are fine-tuned with the training data for that level. HFT-CNN obtained Micro-F1 and Macro-F1 scores of 80.29 and 51.40 respectively on the RCV1-V2 dataset.

Banerjee *et al.* [3] proposed the Hierarchical Transfer Learning (HTrans) model which is a LCN approach that utilises transfer-learning in a top-down fashion. Each of the classifiers comprise a GRU model with an attention mechanism and a FCL with a sigmoid activation function. These classifiers determine whether or not an instance belongs to the node associated with the classifier. The parameters of each classifier are initialised with the parameters of its parent classifier except for the final FCL which is reinitialised and the embedding layer which is frozen after the first level. The rationale behind this architecture is similar to the HFT-CNN approach [76] where the parameters of parent classifiers serve as a good starting point for children node classifiers which are fine-tuned on more specific data. HTrans outperformed HFT-CNN by obtaining Micro-F1 and Macro-F1 scores of 81.51 and 58.49 respectively on the RCV1-V2 dataset.

In 2022, Ma *et al.* [53] proposed the Hybrid Embedding-based text representation for Hierarchical Multi-label Text Classification (HE-HMTC) approach. HE-HMTC uses a LCL approach where a classifier at a specific level leverages the previous level's prediction to improve classification performance. First, the HE-HMTC approach obtains representations of the hierarchical class structure which combine the structural and semantic information of the class nodes. The hierarchical class structure is encoded through the Structural Deep Network Embedding approach [87] which obtains embeddings for each node that capture the relationships between the nodes in the hierarchy. Furthermore, they use the class name descriptions to obtain semantic embeddings of the classes by averaging the word2vec embeddings of the class name tokens. They concatenate the two different embeddings, which capture the hierarchical and semantic information of the classes respectively, to get the hybrid embedding for each node in the hierarchy.

HE-HMTC obtains a vector representation of a text document by converting the document tokens to their word2vec embeddings and passing the embeddings through a GRU model. The classifier at a specific level leverages the previous level predictions by concatenating the hybrid class embedding from the previous level's classification result to the text representation. Finally, the concatenated feature representation is passed to a FCL with a softmax layer to perform multi-class classification for the particular level. HE-HMTC outperformed the HDLTex and HFT-CNN models by obtaining an accuracy of 78.51 on the WOS dataset.

3.3 Global Classifier Approaches

Global classifier approaches attempt to leverage the hierarchical structure of the class taxonomy by incorporating the hierarchical information into a single model. Recently proposed HTC approaches typically use the global classification paradigm [10, 16, 20, 32, 54, 62, 89, 90, 92, 100] and this paradigm is also the focus of this thesis. Furthermore, the recently proposed global approaches generally report their results on the same three benchmark HTC datasets (WOS, RCV1-V2, and NYT) with the same preprocessing steps and dataset splits. Therefore, these approaches can be easily and fairly compared to facilitate further research in the field of HTC. Many global approaches have been proposed which use a diverse range of techniques to incorporate the class hierarchy information into the classification model. However, we focus on the most recent and best-performing approaches in this chapter since they are the most relevant to this thesis.

Some of the earlier proposed global methods used various techniques such as recursive regularisation [20], reinforcement learning [54], meta-learning [92], and capsule networks [62] to leverage the hierarchical class structure information in a single classification model. However, more recent work has shown that encoding the hierarchical class structure through a GNN architecture and incorporating this information into the classification process can significantly improve performance [10, 16, 32, 89, 90, 100].

In 2020, Zhou *et al.* [100] proposed the Hierarchy-Aware Global Model (HiAGM) which comprises two different approaches for incorporating the class hierarchy information into the classification model. Both of their approaches use pre-trained GloVe word embeddings and obtain vector representations of text documents by passing the word embeddings through an LSTM followed by convolutional and max-pooling layers. They propose two methods for using the document representations along with a GCN to solve the HTC problem.

Their first approach, HiAGM multi-label attention (HiAGM-LA), randomly initialises embeddings for each class in the hierarchy and passes the class hierarchy through a GCN which outputs an embedding for each class. These class embeddings capture the hierarchical position of each class through the aggregation of class embeddings performed by the GCN. Therefore, the class embeddings are used as query vectors in an attention mechanism to determine which features of a document are most relevant for each of the classes. The attention mechanism obtains a weighted average of the document representations and passes the result to a FCL which performs the final classification.

The second method, HiAGM text feature propagation (HiAGM-TP), passes the document representation through a FCL which reshapes it to form the node inputs to the GCN. These node representations are passed through the GCN which provides hierarchy-aware text features for each node. Therefore, each of the node features form a label-specific representation of the document which captures the relative hierarchical position of that node. These node features are passed to a FCL which obtains the final classifications by predicting the confidence scores of the document belonging to each of the classes. Zhou *et al.* show that HiAGM-TP outperforms HiAGM-LA as well as previous state-of-the-art approaches with Micro-F1 scores of 85.82, 83.96, and 74.97 and Macro-F1 scores of 80.28, 63.35, and 60.83 on the WOS, RCV1-V2, and NYT datasets respectively.

In 2021, Chen *et al.* [10] proposed the Hierarchy-aware label semantics Matching network (HiMatch) which formulates HTC tasks as a semantic matching problem. HiMatch encodes a text document by converting the words to pre-trained GloVe word embeddings and passing the embeddings through a GRU model, followed by convolutional and max-pooling layers. The document encoding is projected through a FCL to form the node inputs of a GCN model which models the structure of the class hierarchy. Therefore, the GCN combines the document representation with the hierarchical class information to form hierarchy-aware document representations for each class as in the HiAGM-TP approach. Furthermore, their approach creates hierarchy-aware class embeddings by initialising them with the average of the class name GloVe embeddings and similarly passing them through a GCN.

HiMatch projects the hierarchy-aware document and class embeddings into a joint embedding space through separate FCLs in order to perform the semantic matching task, i.e., identifying embeddings which are semantically similar. The model measures the difference between the embeddings and tries to minimise the difference for correct matches while penalising incorrect matches which have small distances between them in the embedding space. Furthermore, the model uses hierarchy-aware penalties to encourage the closest semantic matching between the text representation and correct fine-grained nodes, followed by correct coarse-grained nodes and incorrect siblings of fine-grained nodes. Lastly, other incorrect nodes are penalised to be semantically the furthest away in the joint embedding space. HiMatch outperformed the previous state-of-the-art HiAGM-TP model by obtaining Micro-F1 scores of 86.20 and 84.73 and Macro-F1 scores of 64.11 and 80.53 on the WOS and RCV1-V2 datasets respectively.

Wang *et al.* [89] proposed the Hierarchy Guided Contrastive Learning (HGCLR) HTC approach in 2022. HGCLR uses a contrastive learning approach to embed the hierarchical class structure information into a PLM. HGCLR encodes the class hierarchy with a Graphormer model [98], where the node embeddings for each class are initialised as the sum of the class embedding, which is randomly initialised and learnt during training, and the name embedding, which is the average of the BERT token embeddings of the class name. The node embeddings are passed through the Graphormer to obtain class embeddings which capture the hierarchical position information of the classes. Furthermore, they encode the tokens of a document by passing them through a pre-trained BERT model to obtain the final hidden state for each token.

HGCLR uses the text and hierarchy-aware class representations to generate positive examples, i.e., instances which have correctly assigned classes, by removing a number of original tokens while retaining the correct classes. The model determines the most important tokens for a particular class through an attention mechanism guided by the class representation obtained by the Graphormer. Therefore, for a particular class, the most important tokens of the text document can be sampled to form a positive example that contains very little noise. The positive example is passed to BERT and it is assumed that it maintains its correct classes such that the classification loss of the positive sample guides the learning of the Graphormer and the positive sample generation.

These pairs of text and positive example representations are used to perform contrastive learning [22] by calculating a contrastive learning loss which penalises pairs that are incorrectly similar in the embedding space to guide the training of the model. The rationale behind this design decision is that the representations of the text and its positive example should be close in the embedding space while examples from different pairs should be further apart. Therefore, the model tries to minimise the loss of classification when using the text representations, the loss of classification when using the generated positive samples, and the contrastive learning loss. During inference, the model passes the final hidden state of the `[CLS]` token through a FCL with a sigmoid activation function to obtain the confidence scores for each of the possible classes. HGCLR outperformed previous state-of-the-art models such as HiAGM and HiMatch on the WOS, RCV1-V2, and NYT datasets with Micro-F1 scores of 87.11, 86.49, and 78.86 and Macro-F1 scores of 81.20, 68.31, and 67.96 respectively.

The Hierarchy-Aware T5 model with Path-Adaptive Attention Mechanism (PAAMHiA-T5) was proposed by Huang *et al.* [31] in 2022. Their approach uses the T5 language model [67] which has a standard transformer architecture, i.e., a stack of transformer encoder and decoder modules as depicted in Figure 2.8. Therefore, the T5 model has a seq2seq architecture which is designed to transform any NLP task into a text-to-text task through the encoder and decoder modules.

First, PAAMHiA-T5 passes the text token sequence to the T5 encoder to obtain the semantic feature representation of the token sequence. Furthermore, the model transforms the hierarchical class set into a class token sequence by representing intra-level connections between classes with "_" and inter-level relationships with "/". This class token sequence allows the model to incorporate the class hierarchy into the T5 model by forming the input to the decoder. The decoder passes the class sequence through the masked multi-head self-attention layer to obtain class representations which capture the hierarchical relationships across the different levels. These hierarchy-aware class representations are used as the queries in the encoder-decoder multi-head self-attention mechanism where the encoded text representation forms the keys and values. Therefore, the class representations are used to guide the decoder in determining which features of the text representations are most relevant for classification. The output of the decoder is passed to a FCL with a softmax function which forms the output of the model. Furthermore, the model uses a path-adaptive attention mechanism which allows the model to dynamically focus on the path of the currently generated class such that noise from other paths may be ignored. This method uses an attention mask in the decoder and a regularisation function during the training phase to guide the model to focus on parent classes in the current path while penalising other paths.

PAAMHiA-T5 outperformed the HiAGM and HiMatch models by obtaining Micro-F1 scores of 90.36, 87.22, and 77.52 and Macro-F1 scores of 81.64, 70.02, and 65.97 on the WOS, RCV1-V2, and NYT datasets respectively. Furthermore, PAAMHiA-T5 outperformed HGCLR in terms of Micro-F1 and Macro-F1 on the WOS (+3.25 and +0.44) and RCV1-V2 (+0.73 and +1.71) datasets. However, it should be noted that the underlying T5 model comprises an encoder and decoder such that it uses twice the number of model parameters as the other PLMs used in this thesis.

At a similar time, Jiang *et al.* [32] proposed the Hierarchy-guided BERT with Global and Local hierarchies (HBGL) approach. The HBGL approach uses BERT as a graph encoder to obtain class embeddings that capture the class hierarchy structure by leveraging the pre-trained knowledge of BERT. The class embeddings are initialised as the average of the BERT token embeddings of the class names and combined with position and segment embeddings. The position embedding IDs are chosen as the hierarchy level of the class in order to incorporate the hierarchical information into the class embedding. Furthermore, the segment embeddings IDs are chosen as 1 to represent the classes such that the BERT model can easily distinguish between classes and text. The class embeddings are passed to BERT with an attention mask that ensures that classes can only attend to their parent or child nodes and themselves. They use these input embeddings and attention masks to fine-tune a BERT model with the MLM task by randomly replacing several class embeddings with a `[MASK]` token and trying to predict these masked classes during training. After the training phase the model is used to obtain hierarchy-aware semantic class embeddings for each class in the hierarchy.

HBGL uses the Local Hierarchy-aware Text Encoder (LHTE) to encode a token sequence while leveraging the hierarchical information from each of the text samples through the class embeddings obtained by the procedure described above. The model obtains the local class hierarchy of a document as a sequence by summing the class embeddings of the associated document at each level of the hierarchy and concatenating the resulting embeddings for each level. Therefore, the local class hierarchy captures the class embeddings of the classes assigned to the document at each level. HBGL modifies the input to the BERT model such that it includes the input text token sequence, the local hierarchy, and a masked local hierarchy which comprises a `[MASK]` token for each level of the class hierarchy. However, the model uses an attention mask in the multi-head self-attention layers to ensure that the input text tokens are unable to attend to the local hierarchy and masked local hierarchy. Furthermore, the attention mask allows the masked local hierarchy to only attend to predicted ancestor level classes in the hierarchy. These inputs are passed to BERT which tries to predict the masked local hierarchy, i.e., the classes in each level. Therefore, HBGL aims to leverage the hierarchy-aware class embeddings to inform the prediction of `[MASK]` tokens for each level which finally form the predicted classes. HBGL outperformed previous state-of-the-art approaches such as HGCLR with Micro-F1 scores of 87.36, 87.23, and 80.47 and Macro-F1 scores of 82.00, 71.07, and 70.19 on the WOS, RCV1-V2, and NYT datasets respectively. HBGL also outperformed PAAMHiA-T5, whose underlying PLM uses twice the number of model parameters, on all of the evaluation metrics apart from the WOS Micro-F1 score.

Wang *et al.* [90] also proposed the Hierarchy-aware Prompt Tuning (HPT) approach in 2022. Instead of applying the standard fine-tuning paradigm to a PLM, they proposed a hierarchy-aware prompt tuning method which has the objective of bridging the gap between the pre-training tasks of PLMs and the downstream HTC task.

Prompt tuning attempts to transform the downstream task into the pre-training task of the PLM to leverage the PLM more effectively during fine-tuning [74]. They focused on the BERT model which uses the MLM pre-training task to mask certain tokens in the input sequence and has the objective of predicting these tokens to drive the training process. Standard fine-tuning approaches for text classification simply use the final hidden state of the classification token (`[CLS]`) to perform classification and drive the fine-tuning process. Alternatively, prompt tuning modifies the input sequence that is passed to the model such that the fine-tuning task of the model is better aligned with the pre-training task. This can be done, for example, by converting the input token sequence $\mathbf{x}$, to "$\mathbf{x}$ is about `[MASK]`", and passing it to the PLM which tries to predict the `[MASK]` token.

To incorporate the class hierarchy through prompt tuning, HPT uses a level-wise prompt tuning approach. Suppose the input text token sequence is given as $\mathbf{x}$ and K is the number of levels in the hierarchical class structure. HPT transforms the input sequence which is passed to BERT to "$\mathbf{x}$ $\texttt{[V}_1\texttt{]}$ `[PRED]` $\cdots$ $\texttt{[V}_K\texttt{]}$ `[PRED]`", where the task is to predict the `[PRED]` tokens as the classes for each level. The prompt tokens ($\texttt{[V}_1\texttt{]} \cdots \texttt{[V}_K\texttt{]}$) represent the prompts for each level in the class hierarchy and they have associated embeddings which are learnt during training.

HPT uses a GAT to aggregate information from the classes in each of the levels and obtain hierarchy-aware prompt embeddings. To obtain the level-wise prompt embeddings they create K virtual nodes ($P_1, \ldots, P_K$) and connect each virtual node P_i to all of the nodes in the i-th level of the class hierarchy. This allows the virtual nodes to capture all required node information of the associated level. The embeddings for the virtual nodes are randomly initialised and the embeddings for the class nodes are initialised by averaging the embeddings of their class name tokens. These nodes and their associated connections are passed through the GAT to obtain the embeddings for each of the virtual nodes which are used as the embeddings of the level-wise prompt tokens.

The modified input sequence is passed through a BERT model which obtains the final hidden states for each of the tokens. The final hidden states of the `[PRED]` tokens are used for classification in each of the associated levels. Furthermore, global classifier approaches typically convert the HTC task into a multiple binary classification problem with a binary-cross entropy loss function. However, this approach ignores the correlation between classes and since MLM is a multi-class classification problem which uses a softmax function and a cross entropy loss function, there is a gap between the pre-training and fine-tuning tasks [90]. Therefore, HPT uses a zero-bounded multi-label cross entropy loss function to bridge this gap. HPT outperformed previous state-of-the-art methods such as HGCLR by obtaining Micro-F1 scores of 87.16, 87.26, and 80.42 and Macro-F1 scores of 81.93, 69.53, and 70.42 on the WOS, RCV1-V2, and NYT datasets respectively. HPT only outperformed HBGL in terms of Micro-F1 on RCV1-V2 (+0.03) and Macro-F1 on NYT (+0.23) but achieved comparable performance across the three datasets apart from the RCV1-V2 Macro-F1 score (−1.54).

Table 3.3 summarises the performance results of the global classifier approaches mentioned above on the three commonly used benchmark datasets. The results show that the state-of-the-art approaches include the PAAMHiA-T5, HBGL, and HPT models. However, as mentioned above, PAAMHiA-T5 uses an underlying PLM with twice the number of model parameters as the other approaches such that they can not be directly compared fairly.

Table 3.3: Summary performance results of global classifier approaches on the three most commonly used benchmark datasets.

Model	Year	WOS		RCV1-V2		NYT	
		Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
HiAGM-LA [100]	2020	84.82	79.51	82.54	61.90	72.50	58.86
HiAGM-TP [100]	2020	85.82	80.28	83.96	63.35	74.97	60.83
HiMatch [10]	2021	86.20	80.53	84.73	64.11	–	–
HGCLR [89]	2022	87.11	81.20	86.49	68.31	78.86	67.96
PAAMHiA-T5 [31]	2022	**90.36**	81.64	87.22	70.02	77.52	65.97
HBGL [32]	2022	87.36	**82.00**	87.23	**71.07**	**80.47**	70.19
HPT [90]	2022	87.16	81.93	**87.26**	69.53	80.42	**70.42**

Chapter 4

Prompt Tuning Discriminative Language Models for Hierarchical Text Classification

4.1 Introduction

As discussed in Chapter 3, many approaches have been proposed which aim to leverage the structured class hierarchy in order to improve performance on HTC tasks. In particular, global classifier approaches have shown to be a promising area for future research with recently proposed HTC approaches focusing on this paradigm to achieve state-of-the-art performance on the benchmark WOS, RCV1-V2, and NYT datasets [31, 32, 90].

Recent global classifier approaches use GNNs to obtain feature representations of the class hierarchy and incorporate these representations into the classification process in various ways [10, 16, 32, 89, 90, 100]. Moreover, state-of-the-art approaches leverage the language understanding capabilities of transformer-based PLMs by using different techniques to fine-tune the PLM on downstream HTC tasks [31, 32, 90].

Wang *et al.* [90] state that even though the vanilla fine-tuning paradigm has proven successful, recent studies indicate that this paradigm may suffer from the gap between the pre-training and fine-tuning phases such that the fine-tuned models are unable to effectively utilise the knowledge of PLMs [12]. The discrepancy between pre-training and fine-tuning arises since fine-tuning tasks are rarely related to the tasks used to pre-train the language model. Prompt tuning aims to reduce this gap by transforming the input sequence such that the downstream task resembles the pre-training task of the PLM [74]. Therefore, as described in Chapter 3, Wang *et al.* [90] proposed the HPT approach which solves HTC tasks through the prompt tuning paradigm for PLMs that use the MLM pre-training task.

Yao *et al.* [96] proposed the Prompt Tuning framework for Discriminative

PLMs (DPT) which applies the prompt tuning paradigm to discriminative PLMs which use the replaced token detection pre-training task [14]. They showed that their approach solved several NLP tasks, including text classification, more effectively than vanilla fine-tuning approaches. However, the DPT approach requires representations of each class to form part of the input sequence which is passed to the PLM. This reduces the number of tokens available for the original input text. Therefore, DPT does not scale to classification tasks with a large number of classes due to the sequence length limits of transformer-based PLMs which typically only allow 512 input tokens.

To the best of our knowledge, discriminative PLMs such as ELECTRA and DeBERTaV3 have not been used to solve HTC tasks. Consequently, the application of the prompt tuning paradigm to these models for HTC tasks has not been investigated previously. Based on the observation that the DPT approach has proven effective for flat text classification tasks and further that the HPT approach has shown that the use of hierarchy-aware prompts achieves impressive results on HTC tasks, we analyse the effectiveness of combining these two approaches in this chapter.

We propose a new approach for solving HTC tasks called Hierarchy-aware Prompt Tuning for Discriminative PLMs (HPTD). HPTD applies the prompt tuning paradigm to discriminative PLMs for HTC tasks by using hierarchy-aware prompts to transform the input sequence to resemble the RTD task used during pre-training. Our approach reduces the gap between the pre-training and fine-tuning phases, thereby leveraging the discriminative PLM more effectively compared to previous approaches. Furthermore, we propose several improvements to the DPT method that allows more space to be used by text tokens in the input sequence. This is achieved by assigning the same position IDs to class representations that belong to the same level of the class hierarchy, as well as using a learnable embedding representation for each class.

We show the effectiveness of our method by achieving state-of-the-art results on two out of three commonly used benchmark datasets. Furthermore, we analyse the robustness of our approach through rigorous experiments which include an ablation study, stability analysis, and performance changes under simulated low-resource settings.

4.2 Background

4.2.1 Vanilla Fine-tuning for Text Classification

Vanilla fine-tuning for text classification typically uses the classification token `[CLS]`, which is prepended to the start of the input sequence, as a representation of the document which is used for classification. The PLM obtains the final hidden representation of the `[CLS]` token and passes it through a FCL which obtains the confidence scores for each of the possible classes. Dodge *et al.* [18] show that the vanilla fine-tuning paradigm can produce highly unstable results, where different seeds which control the weight initialisation and training data orders lead to significant variance in performance. Furthermore, this paradigm typically leads to a gap between the pre-training and fine-tuning phases since the tasks used to pre-train the model are different from the downstream tasks used during fine-tuning. Therefore, the language understanding capabilities of the PLM may not be effectively leveraged to perform the downstream task during fine-tuning [12].

4.2.2 Prompt Tuning for Text Classification

Prompt tuning is a fine-tuning approach which aims to minimise the gap between pre-training and fine-tuning by transforming the input sequence such that the downstream task resembles the pre-training task of the PLM [74]. The rationale behind this approach is that the model is able to utilise the knowledge of the PLM more effectively during the fine-tuning phase which improves performance on downstream NLP tasks.

Prompt tuning approaches have mostly been applied to PLMs which use the MLM pre-training task [23, 65, 74]. For example, they transform the input token sequence $\mathbf{x} = [x_1, \dots, x_T]$, where T is the number of tokens in the sequence, to "$\mathbf{x}$ is about `[MASK]`" and pass it to the PLM which tries to predict the `[MASK]` token as the class that $\mathbf{x}$ belongs to.

Prompt tuning is broadly divided into two categories: hard prompts [74] which select existing tokens from the vocabulary of the PLM to form the prompts that are added to the input sequence, and soft prompts [23, 65] which initialise a continuous vector representation for prompts and learn these representations during training so that manually chosen prompts are not required.

4.2.3 Prompt Tuning for Discriminative Language Models

Yao *et al.* [96] introduced the DPT approach which applies the prompt tuning paradigm to discriminative PLMs to solve NLP tasks such as text classification.

For a flat text classification task, let L be the number of classes such that $\{c_1, \dots, c_L\}$ is the class set and $\{t_1, \dots, t_L\}$ are the associated class name tokens. For a text token sequence $\mathbf{x}$, DPT modifies the input sequence to "$\mathbf{x}$ Class: $t_1, \dots, t_L$", where the model's task is to predict whether each of the class tokens $t_1, \dots, t_L$ are `original` (correct) or were `replaced` (incorrect).

The modified input sequence is passed to the discriminative PLM which obtains the final hidden state for each token. The score for class c_i is calculated as:

$$s_i = 1 - \sigma(\mathbf{W}_{\text{RTD}}\mathbf{h}_{t_i}) \tag{4.2.1}$$

where $\mathbf{h}_{t_i}$ is the final hidden state of token t_i, and $\mathbf{W}_{\text{RTD}}$ is the RTD head, i.e., the final FCL of the PLM that maps the hidden state to an output representing whether the token is `original` or `replaced`. For class names with more than one token, DPT includes each of the tokens in the input sequence and uses the score obtained for the first token as the score for the class as a whole.

4.3 Methodology

In this section we define our HTC approach named Hierarchy-aware Prompt Tuning for Discriminative PLMs (HPTD). Figure 4.1 illustrates the high-level architecture of HPTD.

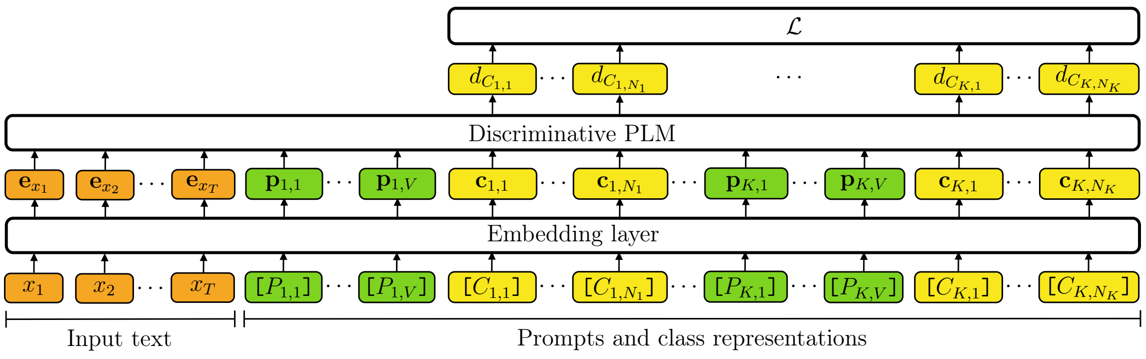

$\mathcal{L}$
$d_{C_{1,1}}$ ··· $d_{C_{1,N_1}}$ ··· $d_{C_{K,1}}$ ··· $d_{C_{K,N_K}}$
Discriminative PLM
$\mathbf{e}_{x_1}$ $\mathbf{e}_{x_2}$ ··· $\mathbf{e}_{x_T}$ $\mathbf{p}_{1,1}$ ··· $\mathbf{p}_{1,V}$ $\mathbf{c}_{1,1}$ ··· $\mathbf{c}_{1,N_1}$ ··· $\mathbf{p}_{K,1}$ ··· $\mathbf{p}_{K,V}$ $\mathbf{c}_{K,1}$ ··· $\mathbf{c}_{K,N_K}$
Embedding layer
x_1 x_2 ··· x_T $[P_{1,1}]$ ··· $[P_{1,V}]$ $[C_{1,1}]$ ··· $[C_{1,N_1}]$ ··· $[P_{K,1}]$ ··· $[P_{K,V}]$ $[C_{K,1}]$ ··· $[C_{K,N_K}]$
Input text
Prompts and class representations

Figure 4.1: The HPTD architecture during training. HPTD modifies the input token sequence of length T (orange) by appending a template that comprises V prompts for each of the K levels in the class hierarchy (green), followed by a representation of each class in the associated level (yellow). The tokens are passed through the discriminative PLM to obtain the output for the class tokens which are used to drive the training process.

4.3.1 Hierarchy-aware Prompts

We follow the same approach as HPT [90] to construct prompts for each of the levels in the hierarchy to incorporate the hierarchical class structure information during the fine-tuning process. Figure 4.2 shows an example of the high-level procedure used to obtain the embeddings of the hierarchical prompts. We construct a graph that represents the K-level hierarchical class structure of the associated task, and initialise the embeddings associated with each node in the graph as the average of the embeddings of their class name tokens. Furthermore, we randomly initialise the embeddings of K virtual nodes $P_i \forall\, i \in \{1, \ldots, K\}$ where each virtual node P_i is connected to each class node at level i of the graph such that each virtual node can capture the information of its associated level. This is achieved by passing the graph structure through a GAT which aggregates the information between the classes in each level to

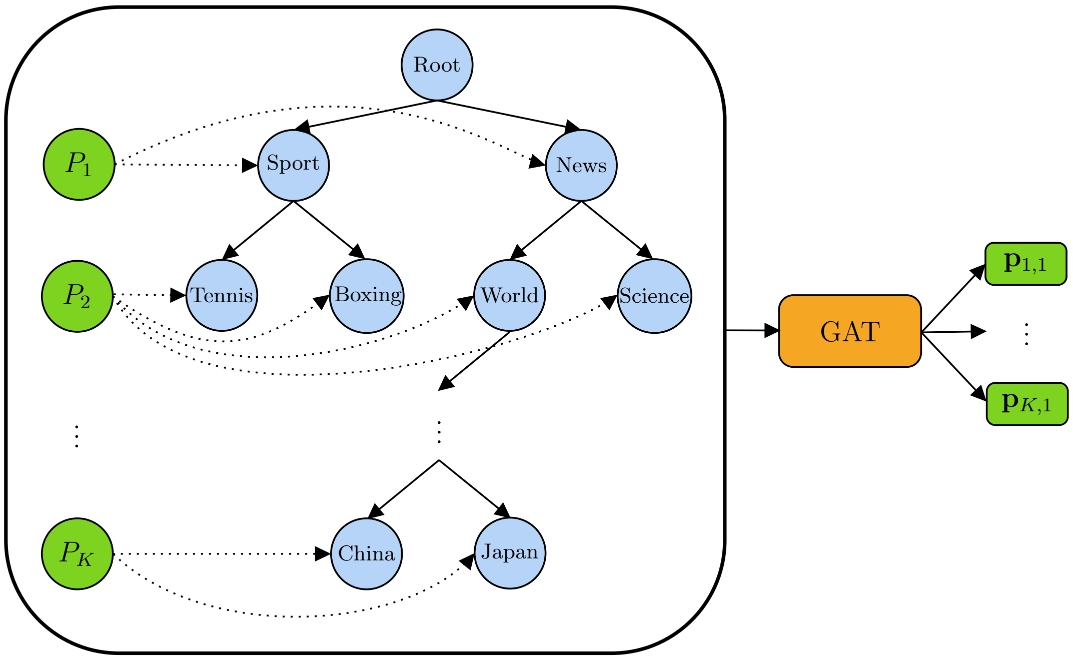


Figure 4.2: High-level procedure used to obtain the hierarchical prompt embeddings. We construct the hierarchical class graph and attach virtual nodes that connect to all of the classes at a particular level. The graph is passed through a GAT which obtains the embeddings for the prompts associated with each level.

the virtual nodes. Finally, the embeddings of the virtual nodes are used as the embeddings of the first prompt at each level ($\mathbf{p}_{1,1} \cdots \mathbf{p}_{K,1}$).

4.3.2 Model Architecture

Suppose we have a hierarchical class structure with K levels, and N_i is the number of classes in level $i \in \{1, \ldots, K\}$ of the hierarchy. Furthermore, we have a text token sequence $\mathbf{x}$, class tokens $[C_{i,j}]$ which represent classes $j \in \{1, \ldots, N_i\}$ at levels i, and corresponding prompt tokens $[P_{i,v}]$ which represent prompts $v \in \{1, \ldots, V\}$, where V is the number of prompts per level. HPTD modifies the input sequence to transform the HTC task such that it resembles the RTD task which predicts each of the class tokens ($[C_{i,j}]$) as `original` or `replaced`. Therefore, the input sequence which is passed to the discriminative PLM is constructed as follows:

$$\begin{aligned} &\mathbf{x}\,[P_{1,1}] \cdots [P_{1,V}]\,[C_{1,1}] \cdots [C_{1,N_1}] \cdots \\ &\cdots [P_{K,1}] \cdots [P_{K,V}]\,[C_{K,1}] \cdots [C_{K,N_K}] \end{aligned} \tag{4.3.1}$$

We omit the `[CLS]`, `[SEP]` and `[PAD]` tokens for sake of clarity. The `[PAD]` tokens are added before the prompt and class tokens for sequences that use less than 512 tokens. This is done to guarantee that the prompt and class tokens always preserve the same positions in the input sequence. The PLM converts each token to its vector representation through the embedding layer to obtain:

$$\mathbf{X}, \mathbf{p}_{1,1}, \ldots, \mathbf{p}_{1,V}, \mathbf{c}_{1,1}, \ldots, \mathbf{c}_{1,N_1}, \ldots, \mathbf{p}_{K,1}, \ldots, \mathbf{p}_{K,V}, \mathbf{c}_{K,1}, \ldots, \mathbf{c}_{K,N_K} \tag{4.3.2}$$

where $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$ are the embeddings of the input text tokens, $\mathbf{p}_{i,v}$ is the prompt embedding for the v-th prompt at level i, and $\mathbf{c}_{i,j}$ is the class embedding for the j-th class at level i. The embeddings of the class tokens are initialised with the average of their class name token embeddings, while the embeddings of the first prompt token at each level $(\mathbf{p}_{1,1} \cdots \mathbf{p}_{K,1})$ are obtained by the procedure described above. These embeddings are passed through the discriminative PLM which obtains the output for each class token as:

$$d_{C_{1,1}}, \ldots, d_{C_{1,N_1}}, \ldots, d_{C_{K,1}}, \ldots, d_{C_{K,N_K}} \tag{4.3.3}$$

where $d_{C_{i,j}}$ is the output score of the j-th class at level i.

The class score that represents the confidence that the text document belongs to the j-th class at level i is calculated as $s_{i,j} = 1 - \sigma(d_{C_{i,j}})$. The rational behind this formulation of the class scores is based on the fact that discriminative PLMs' objective is to assign small scores to `original` tokens and large scores to `replaced` tokens [96]. The model uses these class scores to calculate the classification loss using a binary cross entropy loss function as follows:

$$\mathcal{L} = -\sum_{i=1}^{K} \sum_{j=1}^{N_i} (y_{i,j} \log(s_{i,j}) + (1 - y_{i,j}) \log(1 - s_{i,j})) \tag{4.3.4}$$

where $y_{i,j} \in \{0, 1\}$ is the ground-truth label for the j-th class in level i.

During inference, the class scores are used to predict the multi-hot vector $\mathbf{Y}' = [y'_{1,1}, \ldots, y'_{K,N_K}]$ which represents the class set associated with the text document and is calculated as:

$$y'_{i,j} = \begin{cases} 1, & s_{i,j} \geq \gamma \\ 0, & s_{i,j} < \gamma \end{cases} \tag{4.3.5}$$

where γ is a threshold that determines whether the class is assigned to the document or not.

4.3.3 Adaptions to DPT

As mentioned previously, the major shortcoming of the DPT approach for text classification tasks is that it transforms the input sequence to include the tokens associated with each of the class names in the label set. This significantly reduces the available space for text tokens in the input sequence, given the limited token sequence lengths in PLMs. This renders the approach infeasible for text classification problems with a large number of classes because there is little or no space left for the input text tokens.

However, the only limitation that prevents input sequences from being longer than the model's predefined token sequence length are the position embeddings assigned to tokens. These position embeddings are required since transformer architectures do not naturally capture positional information [84]. Each word is assigned a position ID based on its position in the input sequence. PLMs typically use a maximum sequence length of 512, so position IDs are assigned from 0 to 511 sequentially for each of the tokens in the input sequence. These position IDs are mapped to position embeddings through the position embedding matrix, which is learnt during training and has a corresponding length of 512. Therefore, the PLM does not accept position IDs larger than 511, since it does not have a corresponding entry in the position embedding matrix.

To address this limitation of DPT, we assign the same position IDs to class representations in the same level and use a learnable vector representation for each class.

4.3.3.1 Position IDs

Given the structure of the input sequence formed by our approach, the position ID assignments may be exploited to avoid the limitation of DPT. Since the objective of the model is to determine whether the class representations after a level-wise prompt are `original` or `replaced`, we argue that the positional information of each class in the same level may be reused. Therefore, we assign the same position IDs to all of the class tokens in the same level because these classes have no inherent ordering. This allows HPTD to scale to HTC tasks with much larger hierarchical class structures while maintaining many more input text tokens than the DPT approach. Table 4.1 shows a comparative illustration of the models' encoding efficiencies by using this approach of position ID assignments in HPTD.

4.3.3.2 Soft Class Representations

DPT uses the tokens of the class name descriptions to represent each of the classes in the input sequence. Therefore, for class names that comprise multiple tokens, each of the tokens are added to the input sequence and the first of these tokens is used to obtain the class score. We improve this approach by creating a learnable vector representation for each class token ($\texttt{[}C_{i,j}\texttt{]}$) and initialise its embedding ($\mathbf{c}_{i,j}$) with the average of its token embeddings. This further increases the number of input text tokens that can form part of the input sequence compared to the DPT approach since each class only requires one token.

Table 4.1: Comparison of token space available for input text tokens using the DPT and HPTD approaches for different hierarchical class structures. We assume that the sequence length is limited to 512 tokens and the hierarchical prompts and class representations are included in the input sequence as described in Section 4.3.2, where the number of prompts per level (V) is 1. The columns "Levels" and "Classes" give the number of levels and classes in the class structure, while "Tokens" is the number of tokens that can be used for input sequence text and "%Tokens" is the percentage of the maximum input sequence length (512) that is available for the text tokens to occupy. "Additional tokens" is the number of additional tokens that are passed to the model in HPTD over the 512 token sequence length of DPT. "Ill. Ex." stands for "Illustrative Example" which is an artificial dataset to show the encoding efficiencies for the different approaches.

Dataset	Levels	Classes	DPT		HPTD		
			Tokens	%Tokens	Tokens	%Tokens	Additional tokens
WOS	2	141	369	72.07	508	99.21	+139
RCV1-V2	4	103	405	79.10	504	98.44	+99
NYT	8	166	338	66.02	496	96.88	+158
Ill. Ex.	2	50	460	89.84	508	99.21	+48
Ill. Ex.	2	200	310	60.54	508	99.21	+198
Ill. Ex.	2	800	0	0	508	99.21	+798
Ill. Ex.	8	50	454	88.67	496	96.88	+42
Ill. Ex.	8	200	304	59.38	496	96.88	+192
Ill. Ex.	8	800	0	0	496	96.88	+792

4.4 Experiments

4.4.1 Threshold Selection

The commonly used approach for multi-label text classification tasks is to set the threshold value $\gamma = 0.5$. We compare this approach to three variations for tuning γ using the development sets of the associated datasets. The variations are:

1. A single threshold γ for all classes.
2. A threshold $\gamma_k \, \forall \, k \in [1, K]$ for each level.
3. A threshold $\gamma_l \, \forall \, l \in [1, L]$ for each class.

We use a bootstrapping technique to determine the thresholds for each of these approaches. We randomly sample 10% of instances from the development set 100 times and calculate the Macro-F1 score for each sample for each of the possible thresholds. The threshold for which the highest average Macro-F1 score is achieved over the 100 samples is chosen as the single threshold, or the threshold for the associated level or label.

4.4.2 Implementation Details

We implemented the HPTD approach using PyTorch, PyTorch Lightning, and Hugging Face. For fair comparisons to previous approaches that use BERT models [10, 32, 89, 90], we use the `electra-base-discriminator` and `deberta-v3-base` models from Hugging Face which have the same number of model parameters as the `bert-base-uncased` model used by previous approaches. However, it should be noted that DeBERTaV3 uses a vocabulary size of 128K as opposed to BERT and ELECTRA that only use 30K. We used the Adam optimiser [36] and performed hyperparameter tuning on the learning rate and batch size with possible values of $\{8\text{e-}6, 1\text{e-}5, 1.5\text{e-}5, 2\text{e-}5\}$ and $\{16, 32\}$ respectively. We set the number of prompts per level (V) to 4. For the threshold tuning approaches we considered the set $\{0.2, 0.3, 0.4, 0.5, 0.6, 0.7\}$ as possible values for γ.

We trained the model on the training set for a maximum of 20 epochs and stopped training when the Macro-F1 score on the development set did not increase for 5 consecutive epochs. We chose the model and hyperparameter combination which obtained the highest Macro-F1 score on the development set to evaluate the performance of our models on the test set.

4.4.3 Main Results

In Table 4.2 we present the results of our approach compared to the most recent HTC approaches using the three commonly used benchmark datasets. We report the average performance from three runs with different random seeds while using a fixed threshold (γ) of 0.5 because previous approaches did not tune this threshold. We use ELECTRA and DeBERTaV3 as the discriminative PLMs in our approach and refer to the two models as HPTD-ELECTRA and HPTD-DeBERTaV3 respectively.

Table 4.2: Performance comparisons of the HPTD approaches using the three commonly used benchmark datasets.

Model	WOS		RCV1-V2		NYT	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
HiMatch [10]	86.20	80.53	84.73	64.11	–	–
HGCLR [89]	87.11	81.20	86.49	68.31	78.86	67.96
PAAMHiA-T5 [1] [31]	**90.36**	81.64	87.22	70.02	77.52	65.97
HBGL [32]	87.36	82.00	87.23	**71.07**	80.47	70.19
HPT [90]	87.16	81.93	**87.26**	69.53	80.42	70.42
HPTD-ELECTRA	87.45	81.67	86.30	68.12	80.54	70.66
HPTD-DeBERTaV3	**87.85**	**82.13**	86.25	66.85	**81.45**	**72.40**

[1]Results obtained using twice the number of model parameters as the other approaches.

The results show that HPTD-DeBERTaV3 outperforms HPTD-ELECTRA on the WOS and NYT datasets and achieves comparable performance on the RCV1-V2 dataset. HPTD-DeBERTaV3 generally outperforms HPTD-ELECTRA due to DeBERTaV3's improved performance on the RTD task. These improvements can be attributed to the attention mechanism that calculates the attention weights between tokens through separate matrices on their contents and relative positions, and the gradient-disentangled embedding sharing method which prevents the "tug-of-war" dynamics where the generator and discriminator pull token embeddings in different directions during training.

HPTD-DeBERTaV3 outperforms the current state-of-the-art approaches (bar models with a significantly larger number of model parameters) on the WOS and NYT datasets. Moreover, HPTD-DeBERTaV3 significantly outperforms previous approaches on the NYT dataset with F1 score improvements of 0.98 (Micro) and 1.98 (Macro) percentage points. The NYT dataset has the most complex hierarchical class structure with the deepest hierarchy, which shows that our approach can leverage more complex hierarchical class structures effectively to improve classification performance. However, our approach did not improve on the HBGL and HPT approaches when evaluated on the RCV1-V2 dataset, which has 20,833 train and 781,265 test instances. We hypothesise that our approach struggles to consistently classify classes that are extremely scarce in the training data compared to previous approaches and provide further analysis in Section 4.4.6.

4.4.4 Stability Analysis

Table 4.3 shows the average and standard deviation over the three runs for each dataset to give an indication of the stability of our two models and provide a baseline for future research. These results were obtained using a threshold value of $\gamma = 0.5$.

Table 4.3 shows that even though HPTD-DeBERTaV3 generally outperforms HPTD-ELECTRA, using ELECTRA as the discriminative PLM in our approach improves the stability over multiple runs with different random seeds. Furthermore, the results show that the Macro-F1 scores are generally more inconsistent than Micro-F1 scores across multiple runs.

Table 4.3: The average performance results of evaluating the HPTD approaches over three independent runs. The values in parentheses show the corresponding standard deviations.

Model	WOS		RCV1-V2		NYT	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
HPTD-ELECTRA	87.45 (0.16)	81.67 (**0.12**)	86.30 (**0.17**)	68.12 (**0.97**)	80.54 (**0.05**)	70.66 (**0.14**)
HPTD-DeBERTaV3	87.85 (**0.13**)	82.13 (0.34)	86.25 (0.19)	66.85 (1.33)	81.45 (0.14)	72.40 (0.41)

4.4.5 Ablation Study

To test the effectiveness of our approach, we remove some components of the model and observe the impact that it has on performance. Table 4.4 shows the ablation results for the HPTD-ELECTRA model on the development set of the NYT dataset. We discuss the ablation study on the NYT dataset since it has the deepest and most complex class hierarchy.

By removing the additional prompts, i.e., only using one prompt per level, the Micro-F1 and Macro-F1 scores decrease by 0.27 and 0.47 respectively. This shows that even though the additional prompts leave less space for input text tokens, they can improve performance by providing better context of each level before the associated class representations.

The removal of the hierarchical prompts, i.e., placing all the prompts before the class representations, also reduces the performance of the model in terms of Micro-F1 (-0.14) and Macro-F1 (-0.42) scores. Therefore, we believe that the placement of the prompts before the class representations of their associated level allow the model to leverage the hierarchical class structure more effectively.

Finally, by removing the embeddings obtained by the GAT for the first prompt at each level, i.e., randomly initialising all of the prompts, the Micro-F1 and Macro-F1 both decrease with 0.10.

The ablation results on the WOS and RCV1-V2 datasets are shown in Tables 4.5 and 4.6 respectively and generally show similarly decreasing performance when removing the various components of HPTD.

Table 4.4: Results on the development set of NYT dataset when removing components of the HPTD-ELECTRA model.

Ablation models	Micro-F1	Macro-F1
HPTD-ELECTRA	80.56	71.99
-*r.m.* additional prompts	80.29	71.52
-*r.m.* hierarchical prompts	80.42	71.57
-*r.m.* GAT prompts	80.46	71.89

Table 4.5: Results on the development set of WOS dataset when removing components of the HPTD-ELECTRA model.

Ablation models	Micro-F1	Macro-F1
HPTD-ELECTRA	88.11	81.87
-*r.m.* additional prompts	88.02	81.73
-*r.m.* hierarchical prompts	87.60	81.24
-*r.m.* GAT prompts	87.77	81.64

Table 4.6: Results on the development set of RCV1-V2 dataset when removing components of the HPTD-ELECTRA model.

Ablation models	Micro-F1	Macro-F1
HPTD-ELECTRA	87.12	68.45
-*r.m.* additional prompts	87.38	66.87
-*r.m.* hierarchical prompts	87.19	68.27
-*r.m.* GAT prompts	86.93	68.00

4.4.6 Level-wise Results

Figure 4.3 presents the Micro-F1 and Macro-F1 scores for the classes at each level for the three benchmark datasets.

These figures show that the performance at a certain level is generally directly correlated to the average number of training instances per class at that level. We see that the Micro-F1 and Macro-F1 scores on the WOS dataset significantly decrease from level 1 to level 2 as the average number of training instances per class reduces. A similar trend is seen in the four-level hierarchical class structure of the RCV1-V2 dataset but interestingly the HPTD-ELECTRA model outperforms HPTD-DeBERTaV3 for the level 2 and 3 classes but HPTD-DeBERTaV3 performs significantly better for level 4 which only has a single class. The level-wise results on the NYT dataset show that the performance generally decreases for the levels with fewer training instances, apart from level 8 which has the lowest Micro-F1 and Macro-F1 scores even though it has a higher average number of training instances per class than levels 3 to 7. However, level 8 of the NYT dataset only has two classes which likely leads to greater variance in performance compared to levels with more classes.

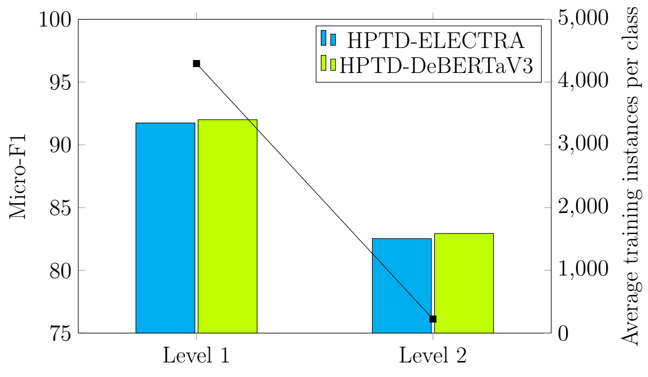
4.4.7 Threshold Selection Results

Table 4.7 presents the results of the HPTD-ELECTRA and HPTD-DeBERTaV3 models for each of the threshold selection approaches.

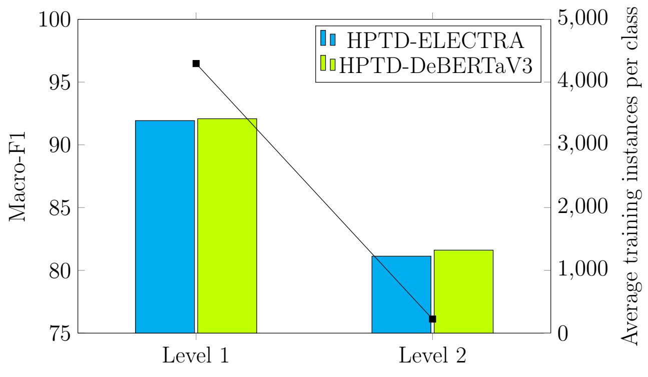
The results show that a fixed threshold of 0.5 consistently achieves the highest Micro-F1 scores across the three datasets for both of our models. We believe that the other threshold selection approaches may overfit the development datasets whereas the static 0.5 threshold generalises better to classes with many instances in the test set.

The per-class threshold selection approach generally obtains the best Macro-F1 scores across the three datasets. We hypothesise that this is because a tuned threshold for each class allows better performance on classes which generally require a lower or higher threshold.

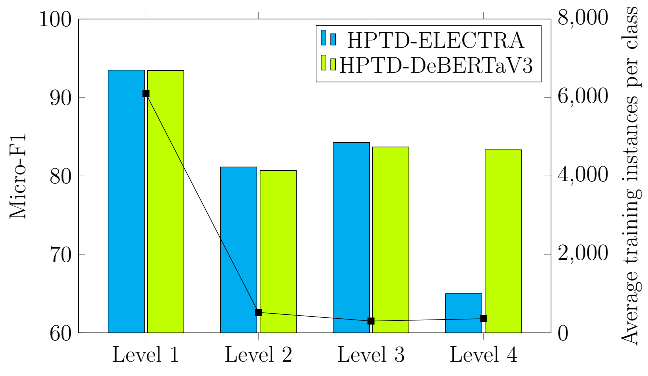
Based on these results, a fixed threshold of 0.5 may be suitable for HTC tasks that require majority classes to be classified accurately whereas a per-



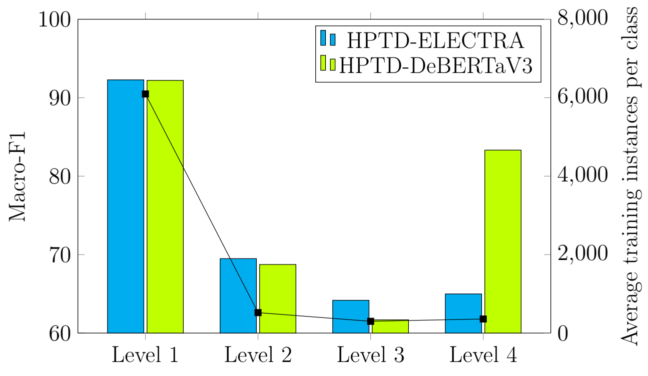
(a) WOS Micro-F1.



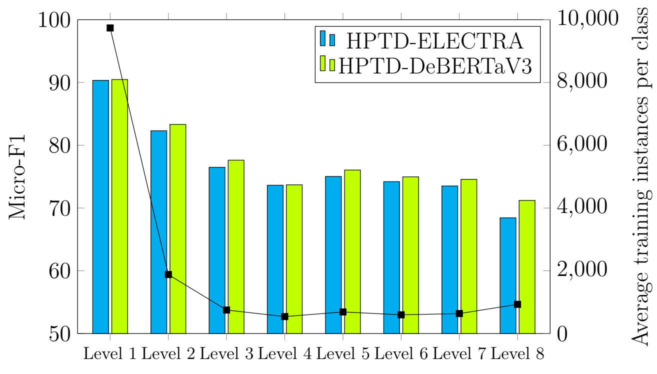
(b) WOS Macro-F1.



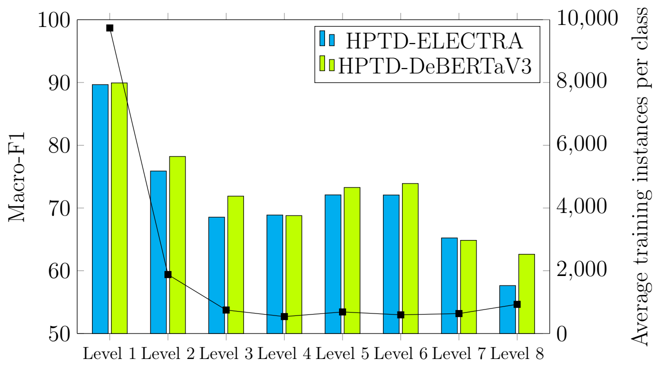
(c) RCV1-V2 Micro-F1.



(d) RCV1-V2 Macro-F1.



(e) NYT Micro-F1.



(f) NYT Macro-F1.

Figure 4.3: Level-wise performance results of the HPTD approaches on the three benchmark datasets. The bar plot gives the F1 scores (left y-axis) for the different approaches at each level of the hierarchy while the line plot shows the average training instances for the classes at a particular level (right y-axis).

class tuned threshold may be beneficial for HTC tasks that require better accuracy across all classes.

Table 4.7: Performance comparisons of the different threshold selection approaches. "0.5" uses a fixed 0.5 threshold for all classes, while "Single" uses a single tuned threshold for all classes. "Level" and "Class" use a tuned threshold for each level and class respectively.

Model	γ	WOS		RCV1-V2		NYT	
		Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
HPTD-ELECTRA	0.5	**87.45**	81.67	**86.30**	68.12	**80.54**	70.66
	Single	87.18	81.75	85.83	68.47	80.37	70.73
	Level	87.18	81.75	85.84	68.52	80.42	70.74
	Class	87.28	**81.93**	86.24	**68.57**	80.50	**70.83**
HPTD-DeBERTaV3	0.5	**87.85**	82.13	**86.25**	66.85	**81.45**	72.40
	Single	87.76	82.45	85.48	67.55	81.19	72.23
	Level	87.81	**82.46**	85.78	67.55	81.26	72.33
	Class	87.49	82.29	86.09	**67.84**	81.40	**72.56**

4.4.8 Low-resource Results

We simulate a low-resource setting to assess the robustness of our approach for HTC tasks when fewer training instances are available. Following Wang *et al.* [90], we randomly sample 10% of the training data to form the training set and keep the other settings the same. Table 4.8 shows the performance results of the HPTD models under the low-resource scenario. Again, the F1 scores are computed as the average over three runs using different random seeds.

HPTD-DeBERTaV3 outperforms HPTD-ELECTRA in the low-resource scenario on the WOS and NYT datasets. Similar to the reasons given in Section 4.4.3, we believe this is due to the improved capabilities of the DeBERTaV3 model which can be attributed to the disentangled attention mechanism and the gradient-disentangled embedding sharing method. However, HPTD-ELECTRA outperforms HPTD-DeBERTaV3 by 3.91 for the Macro-F1 score on the RCV1-V2 dataset, showing that it may be a more suitable approach for low-resource settings with extreme class imbalances.

For both of our models the Micro-F1 does not decrease as significantly as Macro-F1 when using less training data. This shows that Macro-F1 is a stricter metric since it weighs the performance of the classes equally where as Micro-F1 is biased towards the performance of the majority classes.

Table 4.8: Performance results of the HPTD approaches under a low-resource scenario where only 10% of the training data is used. For comparison the achieved results when all training data is used are shown in parentheses.

Model	WOS		RCV1-V2		NYT	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
HPTD-ELECTRA	82.34 (87.45)	74.75 (81.67)	**80.98** (86.30)	**49.07** (68.12)	75.00 (80.54)	61.28 (70.66)
HPTD-DeBERTaV3	**83.47** (87.85)	**75.74** (82.13)	79.42 (86.25)	45.16 (66.85)	**76.18** (81.45)	**63.38** (72.40)

4.5 Conclusion

In this chapter we proposed the Hierarchy-aware Prompt Tuning for Discriminative language models (HPTD) method which effectively classifies text documents into hierarchical class structures. HPTD transforms the input token sequence to include hierarchy-aware prompts followed by class representations for the classes at each level in the hierarchy. Therefore, HPTD injects the HTC task into the RTD pre-training objective used by discriminative language models and leverages the model more effectively by reducing the gap between the pre-training and fine-tuning phases. Furthermore, we proposed improvements to the standard approach of prompt tuning discriminative language models, which allows us to scale HTC tasks to significantly larger class hierarchies. We performed extensive experiments which show that our approach is robust and outperforms current state-of-the-art approaches on two out of three benchmark datasets. Finally, through the analysis of different approaches to tune selection thresholds, we showed that a fixed threshold of 0.5 consistently obtains the best Micro-F1 scores, whereas tuning a selection threshold per class generally achieves the best Macro-F1 results.

Chapter 5

Hierarchical Text Classification using Language Models with Label-wise Attention Mechanisms

5.1 Introduction

As discussed in Section 3.1, several approaches have been proposed which use label-wise attention mechanisms in deep learning architectures to improve classification performance on HTC tasks [4, 47, 50, 59, 80, 86]. These approaches add label-wise attention mechanisms to ANN architectures such as CNNs [47, 59], RNNs [4, 86], and transformer-based PLMs [50, 80] for the task of assigning ICD codes to the MIMIC datasets. The label-wise attention mechanisms obtain label-specific weighted averages of the document features by placing more weight on the most important features for each class separately. Therefore, each class has a representation of the text document which summarises the features of the document that are most important for the classification of that class.

Previous approaches that fine-tune PLMs with label-wise attention mechanisms either use the label-wise attention mechanisms proposed by Mullenbach *et al.* [59] (CAML) or Vu *et al.* [86] (LAAT) but do not compare them and do not propose improvements to them. Therefore, the use of label-wise attention mechanisms to fine-tune PLMs for HTC tasks has not been comprehensively investigated in previous work. Furthermore, these approaches have only been applied to the medical domain and have not been considered for HTC tasks on the WOS, RCV1-V2, and NYT datasets.

In this chapter, we propose a HTC approach which adds a label-wise attention mechanism and classification layer to a PLM and fine-tunes the model for a given downstream HTC task. We compare the performance of several label-wise attention mechanisms to determine the most suitable approach for HTC tasks. These attention mechanisms include a label-wise attention layer with a standard dot product alignment function [59], and an adapted version of the general attention alignment function [86]. Furthermore, we investigate the attention mechanism proposed by Vu *et al.* [86] which splits the attention layers into the different levels of the class hierarchy and uses the information of the parent-level predictions during the prediction task at a certain level. Lastly, we propose an adaptation to this approach which uses the predictions of all ancestor levels during the prediction task at a certain level.

Through comprehensive experiments on three benchmark HTC datasets we show that our proposed label-wise attention mechanism approach generally outperforms the other approaches. We compare the BERT and RoBERTa models as the underlying PLMs in our approach and show that using RoBERTa significantly improves performance on tasks with more complex class hierarchies. Furthermore, we show that our relatively simple approach obtains state-of-the-art performance on two out of three benchmark HTC datasets.

We provide further analysis on the behaviour of our approach by contrasting its per hierarchy level performance against the other approaches and investigating the impact that the per level training set size has on classification performance. Lastly, we investigate the performance of the different label-wise attention mechanisms in a simulated low-resource scenario with only 10% of training instances available and show that using RoBERTa as the underlying PLM significantly improves performance across the three datasets.

5.2 Methodology

Figure 5.1 depicts the high-level architecture of our HTC approach which adds label-wise attention and classification layers to a PLM.

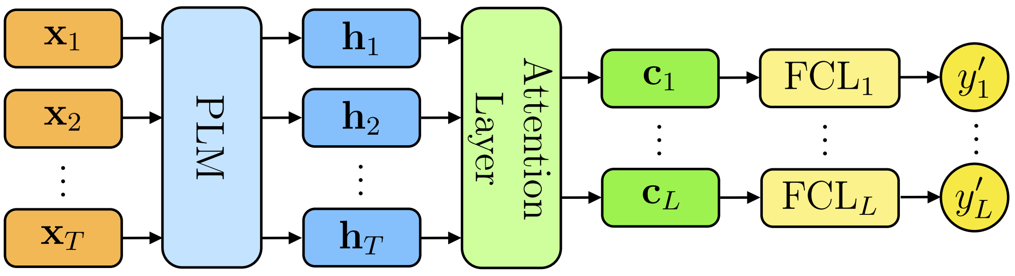


Figure 5.1: Standard label-wise attention model architecture. We pass the text token sequence (orange) through the PLM to obtain the final hidden states for each token (blue). The final hidden states are used in the label-wise attention mechanism which determines a label-specific representation of the document for each class (green) and passes the representations through associated FCLs to obtain a confidence score for each class (yellow).

5.2.1 PLM

Our approach passes the input token sequence $\mathbf{x} = [x_1, \ldots, x_T]$ of length T to the PLM which converts each token through the embedding layer to obtain the token embeddings $\mathbf{E} = [\mathbf{e}_1, \ldots, \mathbf{e}_T]$. We pass these embeddings through the PLM to obtain $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_T] \in \mathbb{R}^{T \times d_h}$, where $\mathbf{h}_i \forall\, i \in \{1, \ldots, T\}$ is the final hidden state of the i-th token and d_h is the dimension size of the hidden states. We compare the BERT and RoBERTa PLMs in our experiments.

5.2.2 Attention Layer

We use the label-wise attention mechanisms proposed by Mullenbach *et al.* [59] (CAML) and Vu *et al.* [86] (LAAT) to convert the concatenated final hidden states ($\mathbf{H}$) to a matrix $\mathbf{C} \in \mathbb{R}^{L \times d_h}$, where L is the number of classes and each row $\mathbf{c}_j \forall\, j \in \{1, \ldots, L\}$ is a label-specific vector representation of the text document. These two attention mechanisms use similar techniques to obtain the attention weights but differ in that they use the dot product attention (DPA) and general attention (GA) functions respectively. Using the DPA approach, the attention weights are calculated as follows:

$$\boldsymbol{\alpha} = \text{softmax}(\mathbf{U}_{\text{DPA}}\mathbf{H}^T) \tag{5.2.1}$$

where $\mathbf{U}_{\text{DPA}} \in \mathbb{R}^{L \times d_h}$ is a weight matrix that is learnt during training and $\boldsymbol{\alpha} \in \mathbb{R}^{L \times T}$ is the attention weight matrix which represents the attention weights of each token for each class. The GA approach extends DPA so that the attention weights are calculated as follows:

$$\mathbf{Z} = \tanh(\mathbf{Q}_{\text{GA}}\mathbf{H}^T) \tag{5.2.2}$$

$$\boldsymbol{\alpha} = \text{softmax}(\mathbf{U}_{\text{GA}}\mathbf{Z}) \tag{5.2.3}$$

where $\mathbf{Q}_{\text{GA}} \in \mathbb{R}^{d_p \times d_h}$ and $\mathbf{U}_{\text{GA}} \in \mathbb{R}^{L \times d_p}$ are weight matrices that are learnt during training and d_p is a chosen hyperparameter which is used to control the dimension sizes of the weight matrices.

The attention weight matrix is used to calculate the label-wise representations of the document as follows:

$$\mathbf{C} = \boldsymbol{\alpha}\mathbf{H} \tag{5.2.4}$$

where the j-th row ($\mathbf{c}_j$) of the matrix $\mathbf{C} \in \mathbb{R}^{L \times d_h}$ represents the information of the document with regard to class j.

5.2.3 Output Layer

The output layer uses the label-wise document representations from the $\mathbf{C}$ matrix and passes each row through a separate FCL with a sigmoid activation function (σ). Therefore, the confidence score of the document belonging to class j is calculated as follows:

$$y'_j = \sigma(\mathbf{w}_j \cdot \mathbf{c}_j + b_j) \tag{5.2.5}$$

where $\mathbf{c}_j$ is the j-th row in $\mathbf{C}$. We pack the weight vectors $\mathbf{w}_1, \dots, \mathbf{w}_L$ and bias units $b_1, \dots, b_L$ into a weight matrix $\mathbf{W} \in \mathbb{R}^{L \times d_h}$ and bias vector $\mathbf{b}_W \in \mathbb{R}^L$ to compute the predictions of all classes as follows:

$$\mathbf{y}' = \sigma\big((\mathbf{W} \otimes \mathbf{C})\mathbf{1} + \mathbf{b}_W\big) \tag{5.2.6}$$

where $\otimes$ is the element-wise multiplication operator and $\mathbf{1} \in \mathbb{R}^{d_h}$ is a vector of ones which is used to obtain the sums over the hidden dimensions.

These confidence scores are used during training to minimize the binary cross-entropy loss which is calculated as:

$$\mathcal{L} = -\sum_{j=1}^{L}\big(y_j \log(y'_j) + (1 - y_j)\log(1 - y'_j)\big) \tag{5.2.7}$$

where $y_j \in \{0, 1\}$ is the ground truth label for class j.

During inference, the confidence scores are used to predict the multi-hot vector $\mathbf{Y}' = [Y'_1, \dots, Y'_L]$ which represents the class set associated with the text document and is calculated as:

$$Y'_j = \begin{cases} 1, & y'_j \geq \gamma \\ 0, & y'_j < \gamma \end{cases} \tag{5.2.8}$$

where γ is a threshold that determines whether a class j is assigned to the document or not.

5.2.4 Hierarchical Attention Mechanisms

We also use the label-wise attention mechanism proposed by Vu *et al.* [86] for the JointLAAT model which leverages the parent-level predictions and splits the attention layers for the levels of the class hierarchy. Figure 5.2 illustrates the high-level architecture of this approach which we refer to as the hierarchical label-wise attention (HLA) mechanism. Suppose we have a structured class hierarchy with K levels, and N_k is the number of classes in level $k \in \{1, \dots, K\}$. HLA obtains the label-wise representation of the document for each class $j \in \{1, \dots, N_1\}$ in the first level by passing the concatenated hidden states ($\mathbf{H}$) through the attention layer associated with level 1 (Attention$_1$) to obtain $\mathbf{C}_1 = [\mathbf{c}_{1,1}, \dots, \mathbf{c}_{1,N_1}]$ where $\mathbf{c}_{1,j}$ is the label-specific representation of the document for the j-th class in level 1.

HLA passes the level 1 class representations through their associated FCLs to obtain the level 1 predictions $\mathbf{y}'_1 = [y'_{1,1}, \dots, y'_{1,N_1}]$, where $y'_{1,j}$ is the confidence score of the j-th class in level 1. The predictions are passed through a

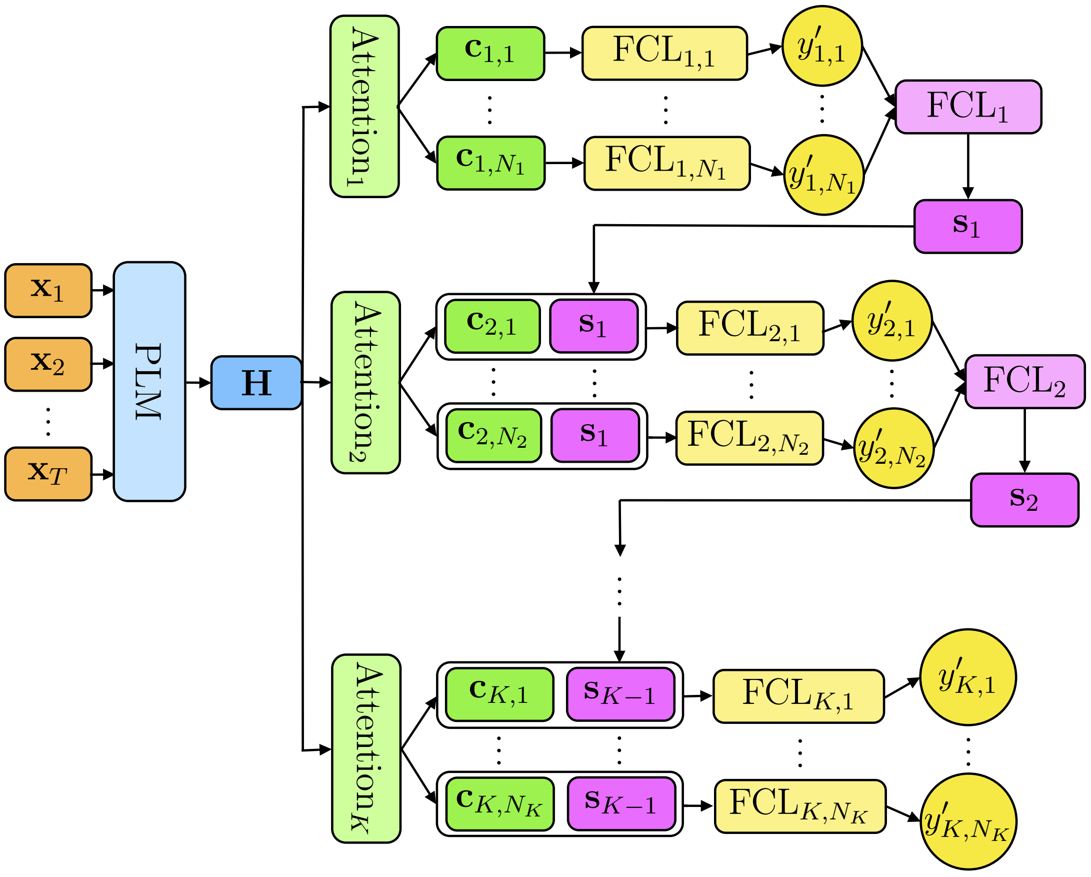


Figure 5.2: Hierarchical label-wise attention model architecture. We pass the input token sequence (orange) through the PLM to obtain the concatenated final hidden states (blue). These are passed to level-specific attention layers to obtain the label-specific representations of the token sequence for each level (green). The predictions of the first-level classes (yellow) are passed through a FCL to obtain a representation of the output (purple) which is concatenated to the label-specific vectors of level two. We repeat this process of using the previous level predictions until the final level of the hierarchy is reached.

FCL with a sigmoid activation function to obtain a vector representation of the predictions at level 1 as:

$$\mathbf{s}_1 = \sigma(\mathbf{y}'_1 \mathbf{F}_1) \tag{5.2.9}$$

where $\mathbf{F}_1 \in \mathbb{R}^{N_1 \times d_k}$ is a weight vector and d_k is a hyperparameter which controls the dimension size of the prediction vector representation. HLA obtains the label-specific vectors for the second-level ($\mathbf{C}_2$) from the associated attention layer (Attention$_2$) and concatenates the vector $\mathbf{s}_1$ to each label-specific vector in level 2 as follows:

$$\mathbf{D}_2 = [Concat(\mathbf{c}_{2,1}, \mathbf{s}_1), \ldots, Concat(\mathbf{c}_{2,N_2}, \mathbf{s}_1)] \tag{5.2.10}$$

where *Concat* concatenates the vectors and $\mathbf{D}_2 \in \mathbb{R}^{N_2\times(d_h+d_k)}$ contains the label-wise representations of the document for the second-level classes. The rows of $\mathbf{D}_2$ are passed through their associated FCLs to obtain the predictions for the second level $\mathbf{y}'_2 = [y'_{2,1}, \ldots, y'_{2,N_2}]$. These predictions are used to determine the level 2 prediction vector representation ($\mathbf{s}_2$) which is concatenated to the label-wise representations of the third-level classes ($\mathbf{C}_3$). This procedure is continued until the final level K is reached and the predictions for each level are combined to form the final prediction scores for the document.

We extend this approach by concatenating all of the ancestor-level prediction representations to the label-specific representations of the current level. Therefore, for the third level of the class hierarchy, the label-wise representations of the document is calculated as:

$$\mathbf{D}_3 = [Concat(\mathbf{c}_{3,1}, \mathbf{s}_1, \mathbf{s}_2), \ldots, Concat(\mathbf{c}_{3,N_3}, \mathbf{s}_1, \mathbf{s}_2)] \tag{5.2.11}$$

This allows the current-level predictions to leverage the information of all higher-level predictions, as opposed to only the parent-level predictions. We refer to this approach as global hierarchical label-wise attention (GHLA) due to the use of all ancestor-level predictions.

5.2.5 Learning Rate Adaptions

To effectively fine-tune the PLM with the added label-wise attention and classification layers, we use three learning rate adaption techniques which include: learning rate warmup, learning rate decay, and layer-wise learning rate decay.

The learning rate warmup linearly increases the learning rate from approximately 0 to the assigned learning rate (η) within a number of steps during the first epoch of training. This prevents the model from overfitting on the first few steps of the training process by gradually increasing the learning rate. The number of steps used to linearly increase the learning rate is calculated by multiplying the total number of steps in an epoch with a warmup fraction λ.

The learning rate decay is applied after the learning rate warmup steps and linearly decreases the learning rate based on the current epoch number (v) as follows:

$$\eta_v = \eta \times \max\left((1 - \frac{v-1}{10})\,,\, 0.1\right) \tag{5.2.12}$$

where η_v is the learning rate for epoch v. Therefore, the learning rate multiplication factor is linearly decreased from 1 to 0.1 over 10 epochs after which it stays constant at 0.1. We use this approach, as opposed to linearly decreasing the multiplication factor to 0, to allow the model to train for an extra few epochs if it improves.

The layer-wise learning rate decay technique exponentially reduces the initial learning rate from the highest to the lowest layer of the model, where the highest and lowest layers are the output and input layer respectively. Suppose we have an assigned learning rate of η and a decaying fraction of β. The initial learning rate for the r-th highest layer of the model is calculated as:

$$\eta^{(r)} = \eta \times \beta^{r-1} \tag{5.2.13}$$

In the context of our model architecture, we group the label-wise attention and classification layers to form the top layer with the initial learning rate η. Furthermore, the layers of the PLM are split into the different transformer-based encoder layers and the embedding layer. These layer-wise learning rates use the same warmup and decaying strategies during training based on their initial learning rates. The reasoning behind the layer-wise learning rate decay approach is that the lower layers of the PLM generally encode high-level or general information while the higher layer representations are more specific to the pre-training task of the PLM [99]. Therefore, by decreasing the learning rates from the highest to the lowest layers of our model, we are able to leverage the language understanding capabilities of the PLM while allowing the task-specific layers to be learnt more effectively.

5.3 Experiments

5.3.1 Implementation Details

We implemented our approaches with the PyTorch, PyTorch Lightning, and Hugging Face libraries. We compared BERT and RoBERTa as the PLM in our models and used the `bert-base-uncased` and `roberta-base` models from Hugging Face.

We used the Adam optimiser [36] and performed hyperparameter tuning on the initial learning rate (η) and batch size with possible values of $\{7.5\text{e-}5, 1\text{e-}4, 2\text{e-}4, 3\text{e-}4\}$ and $\{16, 32\}$ respectively. We chose the learning rate warmup and layer-wise learning rate decay fractions as $\lambda = 0.1$ and $\beta = 0.8$ respectively. Furthermore, we tuned the value of d_p with possible values of $\{256, 512\}$ and chose $d_k = 128$ along with a threshold $\gamma = 0.5$. We used a maximum of 12 epochs and stopped training if the harmonic mean between the Micro-F1 and Macro-F1 on the development set did not increase for 5 consecutive epochs. For the hierarchical attention mechanisms we use GA as the attention layers associated with each level.

We trained each model on the training set and chose the hyperparameter combination which obtained the highest harmonic mean between the Micro-F1 and Macro-F1 scores on the development set. The chosen models were retrained with three random seeds and evaluated on the test set. Therefore, for all experiments in the following subsections, we report the average over three runs with different random seeds.

5.3.2 Main Results

Table 5.1 presents the results of our approaches compared to the recently proposed HTC approaches. We compare the different attention mechanisms with a BERT PLM and select the best performing approach and substitute RoBERTa as the PLM for comparison purposes.

Table 5.1: Performance comparisons of the approaches proposed in this chapter using the three commonly used benchmark datasets.

Model	WOS		RCV1-V2		NYT	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
HiMatch [10]	86.20	80.53	84.73	64.11	–	–
HGCLR [89]	87.11	81.20	86.49	68.31	78.86	67.96
PAAMHiA-T5[1] [31]	**90.36**	81.64	87.22	70.02	77.52	65.97
HBGL [32]	**87.36**	**82.00**	87.23	**71.07**	80.47	70.19
HPT [90]	87.16	81.93	87.26	69.53	80.42	70.42
DPA_{BERT}	87.13	81.48	87.07	68.45	79.67	68.27
GA_{BERT}	87.05	81.46	86.88	69.11	80.06	68.56
HLA_{BERT}	87.17	81.55	86.71	68.45	79.60	68.06
$\text{GHLA}_{\text{BERT}}$	87.17	81.55	87.19	68.62	79.67	68.67
$\text{GHLA}_{\text{RoBERTa}}$	87.00	81.44	**87.78**	70.21	**81.41**	**72.27**

[1]Results obtained using twice the number of model parameters as the other approaches.

Our results show that the DPA and GA approaches generally perform similarly across the three benchmark dataset. However, GA shows improvements over DPA in terms of Macro-F1 on the RCV1-V2 (+0.66) and NYT (+0.29) datasets. Our proposed GHLA approach outperforms the DPA and HLA approaches on all three datasets. It should be noted that the results on the WOS dataset for HLA and GHLA approaches are the same because the WOS class hierarchy only comprises two levels. Furthermore, GHLA outperforms the GA approach on all of the performance measures except for the RCV1-V2 Macro-F1 score and the NYT Micro-F1 score. We hypothesise that GHLA generally outperforms the other label-wise attention mechanisms because it is able to leverage the information of all the ancestor level predictions by splitting up the attention layers at each level. Therefore, GHLA is able to focus on the predictions at a particular level while ignoring classes in lower levels and using the predictions of previous levels to improve performance on HTC tasks.

We select GHLA since it is the best performing label-wise attention mechanism and use RoBERTa as the PLM to compare the difference in performance when using the two PLMs. The results show that the RoBERTa PLM significantly improves the results on the RCV1-V2 and NYT datasets in terms of Micro-F1 (+0.59 and +1.74 respectively) and Macro-F1 (+1.59 and +3.60

respectively). Furthermore, GHLA$_{\text{RoBERTa}}$ improves on state-of-the-art approaches on RCV1-V2 in terms of Micro-F1 (+0.52) and NYT in terms of Micro-F1 (+0.94) and Macro-F1 (+1.85). The NYT dataset has the most complex class hierarchy while the RCV1-V2 class hierarchy is also more complex than WOS. Therefore, due to the significant performance improvements on the NYT dataset particularly, we hypothesise that using RoBERTa as the PLM allows our GHLA approach to more effectively handle complex class hierarchies in general.

5.3.3 Stability Analysis

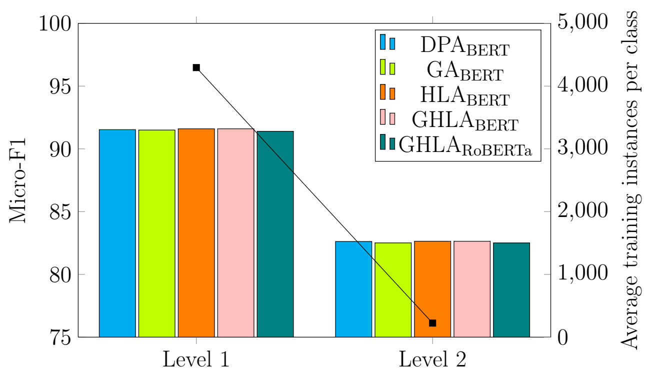
Table 5.2 presents the results of our proposed approaches with standard deviations over the three runs with different seeds. The results show that the DPA and GHLA approaches produce the most stable results on the WOS and RCV1-V2 datasets respectively. Furthermore, the HLA approach has a significantly higher standard deviation on the RCV1-V2 dataset compared to the other label-wise attention mechanisms. Lastly, the results show that the Macro-F1 metric generally produces higher deviation across the multiple runs than the Micro-F1 scores.

Table 5.2: The average performance results of evaluating the approaches proposed in this chapter over three independent runs. The values in parentheses show the corresponding standard deviations.

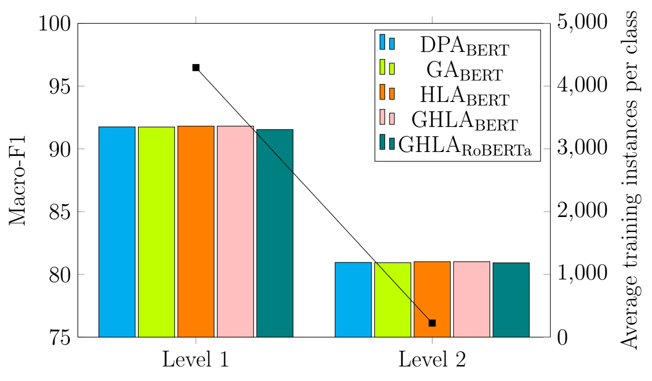
Model	WOS		RCV1-V2		NYT	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DPA$_{\text{BERT}}$	87.13 (**0.09**)	81.48 (**0.13**)	87.07 (0.18)	68.45 (0.22)	79.67 (0.11)	68.27 (0.26)
GA$_{\text{BERT}}$	87.05 (0.17)	81.46 (0.23)	86.88 (0.07)	69.11 (0.04)	80.06 (0.09)	68.52 (0.25)
HLA$_{\text{BERT}}$	87.17 (0.13)	81.55 (0.14)	86.71 (0.46)	68.45 (1.74)	79.60 (**0.04**)	68.06 (0.30)
GHLA$_{\text{BERT}}$	87.17 (0.13)	81.55 (0.14)	87.19 (**0.04**)	68.62 (**0.03**)	79.67 (0.22)	68.67 (0.33)
GHLA$_{\text{RoBERTa}}$	87.00 (0.16)	81.44 (0.23)	87.78 (0.05)	70.21 (0.39)	81.41 (0.09)	72.27 (**0.23**)

5.3.4 Level-wise Results

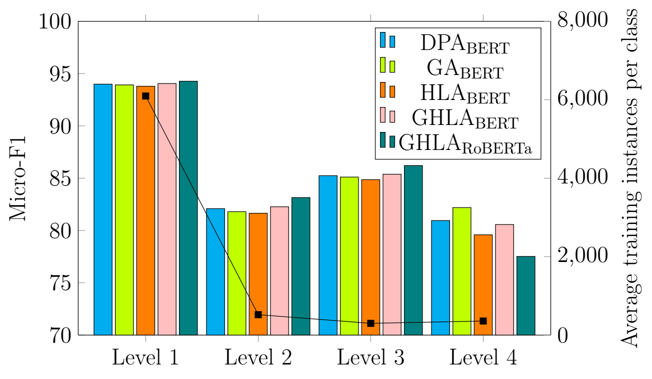
Figure 5.3 presents the Micro-F1 and Macro-F1 scores for the classes at each level for the three benchmark datasets. Our results show that the average number of training instances per class at a certain level generally has a direct correlation with the performance at that level. On the WOS dataset we see that the Micro-F1 and Macro-F1 scores are significantly higher for the first-level classes than the second-level classes. We also see that the average number of training instances are much larger for level 1 classes (4295) than level 2 (224). Similarly, the Micro-F1 and Macro-F1 scores on the NYT dataset show how the average number of training instances per level relates to the performance differences for the classes at each level. Furthermore, the results on the RCV1-V2 and NYT datasets show that the GHLA$_{\text{RoBERTa}}$ model is generally able to leverage the class hierarchy more effectively to outperform the other approaches particularly on the levels with fewer training instances.



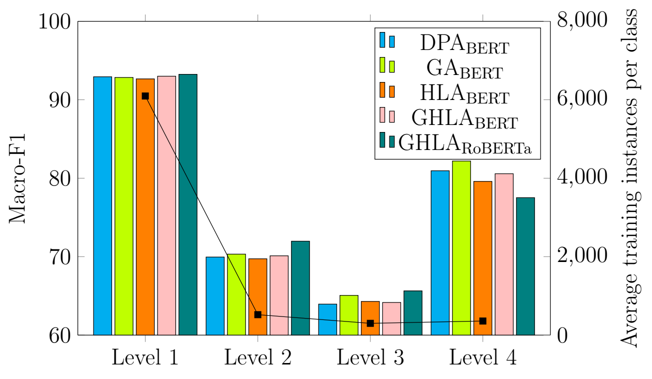
(a) WOS Micro-F1.



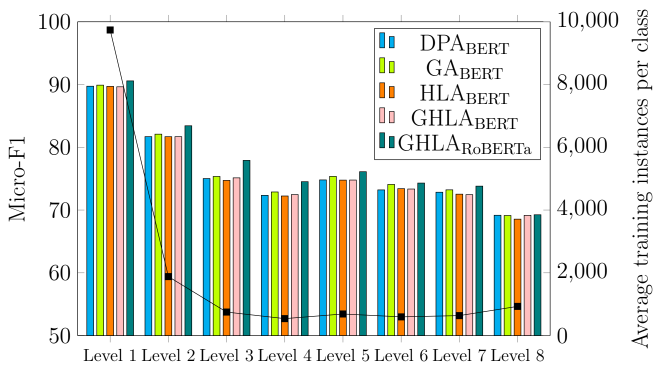
(b) WOS Macro-F1.



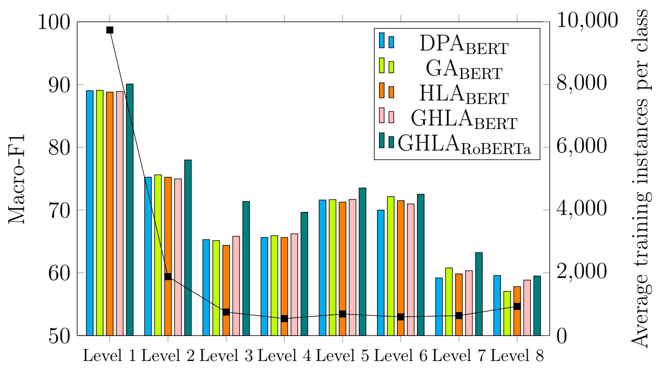
(c) RCV1-V2 Micro-F1.



(d) RCV1-V2 Macro-F1.



(e) NYT Micro-F1.



(f) NYT Macro-F1.

Figure 5.3: Level-wise performance of the approaches proposed in this chapter on the three benchmark datasets. The bar plot gives the F1 scores (left y-axis) for the different approaches at each level of the hierarchy while the line plot shows the average training instances for the classes at a particular level (right y-axis).

5.3.5 Low-resource Results

Table 5.3 presents the results of the approaches for the low-resource setting where only 10% of training data is used as the training set. The results show that the GHLA$_{\text{RoBERTa}}$ model outperforms the other approach on all three benchmark datasets, with significantly higher Macro-F1 scores on the RCV1-V2 and NYT datasets. Furthermore, the DPA mechanism generally outperformed GA, showing that this mechanism may be more suitable for low-resource settings. The HLA mechanism outperformed GHLA on the WOS and RCV1-V2 dataset, but GHLA obtained a significantly higher Macro-F1 score on the NYT dataset, showing that it is able to leverage the more complex class hierarchy to improve performance on scarcely represented classes.

Table 5.3: Performance results of the approaches proposed in this chapter under a low-resource scenario where only 10% of the training data is used. For comparison the achieved results when all training data is used are shown in parentheses.

Model	WOS		RCV1-V2		NYT	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
DPA$_{\text{BERT}}$	79.41 (87.13)	67.51 (81.48)	82.81 (87.07)	52.33 (68.45)	72.29 (79.67)	49.29 (68.27)
GA$_{\text{BERT}}$	79.45 (87.05)	67.50 (81.46)	82.79 (86.88)	49.32 (69.11)	72.54 (80.06)	48.75 (68.56)
HLA$_{\text{BERT}}$	79.53 (87.17)	67.73 (81.55)	82.74 (86.71)	51.87 (68.45)	72.28 (79.60)	45.84 (68.06)
GHLA$_{\text{BERT}}$	78.39 (87.17)	67.03 (81.55)	82.51 (87.19)	50.74 (68.62)	72.42 (79.67)	50.04 (68.67)
GHLA$_{\text{RoBERTa}}$	**79.76** (87.00)	**68.98** (81.44)	**84.45** (87.78)	**55.24** (70.21)	**75.70** (81.41)	**57.85** (72.27)

5.4 Conclusion

In this chapter we proposed a new HTC approach which adds label-wise attention and classification layers to a PLM which is then fine-tuned on a given downstream classification task. It based on an approach which splits the attention layers for each of the levels of the class hierarchy. We suggested an adaption to allow the predictions at a certain level to leverage the information of all ancestor level predictions. We evaluated our suggested adaption by comparing it to different label-wise attention mechanisms approaches through comprehensive experiments on three benchmark datasets. We showed that our proposed label-wise attention mechanism generally outperforms the other approaches. In addition, we also compared BERT against RoBERTa as the underlying PLM and found that using the RoBERTa model significantly improved performance on benchmark datasets that comprise more complex class hierarchies. Finally, we showed that our label-wise attention mechanism with the RoBERTa PLM obtains state-of-the-art performances on two of the three benchmark datasets.

Chapter 6

Combining Language and Topic Models for Hierarchical Text Classification

6.1 Introduction

The approaches proposed in Chapters 4 and 5 use different techniques to fine-tune the parameters of a PLM for HTC tasks. Fine-tuning PLMs is the standard approach used for most NLP tasks since it leverages the language understanding capabilities of the PLM while adjusting the parameters for the specific downstream task. However, PLMs can also be used as feature extraction models by passing a text token sequence through the PLM to obtain semantic and contextual representations for each token in the sequence without updating the parameters of the model. These features can be used to train a new model which aims to learn the functional mapping from the extracted features to the expected output.

Topic models are also used to extract features from a text document by creating abstract topics which are represented by a distribution of words in a corpus of documents. Topic models allow us to represent each document and word as a distribution over the set of abstract topics such that these representations capture the corpus-level topics.

Liu *et al.* [52] proposed a text classification approach which combines the semantic representations extracted from a PLM with the topic representations extracted from a topic model. Their approach used a PLM and a LDA topic model to extract features from text documents to train a CNN classifier. The reasoning behind the combination of these feature extraction models is that the representations obtained from the PLM capture the granular semantic and contextual information of the document while the topic model extracts higher-level "global" information which considers the entire corpus of documents. Their results showed that by combining the features obtained from the two models they were able to improve classification performance on MLTC tasks. They state that combining the semantic- and contextually-aware token representations from the PLM with the "global" topic representations of a LDA topic model improves performance since the features capture different granularities and attributes which can be used to distinguish between the classes of the text documents.

The combination of document representations extracted from a PLM and topic model has not been investigated for HTC tasks in previous work. Therefore, in this chapter, we evaluate the efficacy of using a PLM and a topic model to extract features from text documents which are used to train a classifier model for HTC tasks. The objective of this investigation is to determine whether the improvements obtained by the combination of features which capture different granularities of the document generalises from the text classification approach proposed by Liu *et al.* [52] to HTC tasks. Therefore, we aim to evaluate whether topic models may provide valuable information for distinguishing between different classes as opposed to the standard approach of only using a PLM to extract the semantic representations of the documents.

We propose an approach which uses a BERT PLM and a BERTopic model to extract features from a document which are separately passed through dedicated convolutional layers. Subsequently, the output of the convolutional layers are combined and passed through label-wise attention and classification layers to obtain the final output of the model. We evaluate our proposed model on three benchmark HTC datasets and show that using the topic model features generally worsens the classification performance compared to only using the PLM features. Furthermore, the results show that using a PLM as a feature extractor to train a CNN with label-wise attention consistently performs significantly worse than recently proposed approaches, as well as the other approaches proposed in this thesis.

6.2 Background

Peinelt *et al.* [61] proposed the topic-informed BERT-based model (tBERT) which combines information extracted from a LDA topic model with a BERT PLM for semantic similarity prediction tasks which have the objective of measuring the semantic similarity between text documents. To determine the semantic similarity between two documents, tBERT first concatenates the two documents and passes them through a BERT model to obtain the sentence pair representation from the final hidden state of the `[CLS]` token. Furthermore, tBERT uses a topic model to extract abstract topics from the corpus of documents and represents each document as a distribution over these topics such that they capture the most important topics associated with each document. They concatenate the representations from the BERT and topic models and pass these features through a FCL to obtain the output of the model which represents the semantic similarity of the documents. They show that combining the representations extracted from a topic model and a PLM improved performance over multiple semantic similarity prediction tasks, especially in domain-specific cases.

As described in Section 3.1, Liu *et al.* [52] extended the idea of combining PLMs and topic models by proposing an approach for MLTC tasks which uses a BERT-based PLM and a LDA topic model to obtain "local" semantic and "global" topic information of documents respectively. They showed that combining the local semantic and global topic representations from the PLM and topic models improved classification performance. They state that the combination of these two feature extraction approaches allows the classifier model to leverage the different attributes and granularities of the semantic and topic representations. More specifically, the global topic representations capture higher-level information of the topics contained in the corpus which enables the improved performance.

6.3 Methodology

Figure 6.1 illustrates the high-level architecture of our HTC approach which combines representations from a PLM and a topic model to train a model which comprises convolutional layers followed by label-wise attention and classification layers.

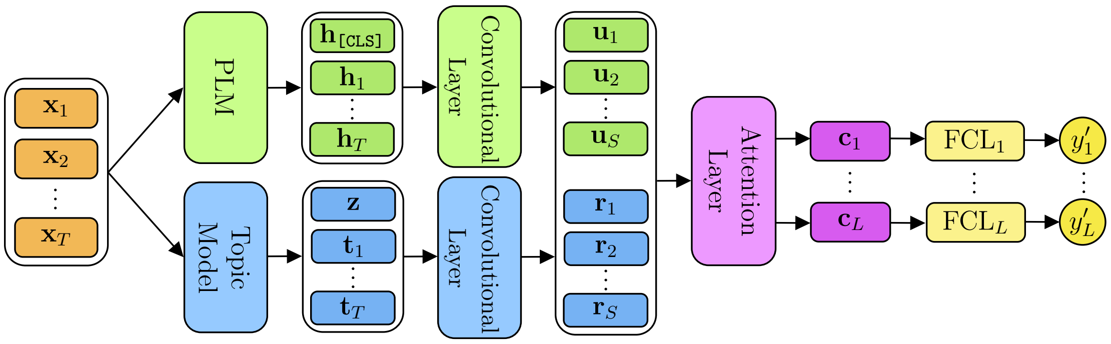


Figure 6.1: Model architecture for extracting features from language and topic models to train a CNN classifier with label-wise attention and classification layers. The input text token sequence (orange) is separately passed through the PLM and topic model to obtain local semantic embeddings (green) and global topic embeddings (blue) for each token in the input sequence and the sequence as a whole. These features are passed through separate convolutional layers whose outputs are combined and passed through a label-wise attention layer which obtains label-specific representations of the document (purple). Finally, these representations are passed through their associated label-wise FCLs which calculate the confidence of the document belonging to each of the possible classes (yellow).

6.3.1 Feature Extraction

Suppose we have a text token sequence $\mathbf{x} = [x_1, \ldots, x_T]$, where T is the number of tokens in the sequence. We extract local semantic and global topic feature vectors for each token in the sequence and the sequence as a whole through a PLM and topic model respectively.

We extract local semantic representations by passing the text token sequence through a PLM to obtain the final hidden states of the token sequence as $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_T] \in \mathbb{R}^{T \times d_h}$, where $\mathbf{h}_t \forall\, t \in \{1, \ldots, T\}$ is the final hidden state for token x_t and d_h is the dimension of the hidden states. These embeddings provide "local" context and semantics of the tokens in a document without considering the other documents in the corpus. Furthermore, the PLM extracts the document-level information through the final hidden state of the `[CLS]` token as $\mathbf{h}_{\texttt{[CLS]}} \in \mathbb{R}^{d_h}$.

The topic model creates d_f abstract topics where each topic is a distribution over the words in the corpus of documents such that it captures the semantics of the topic. We obtain a document-level topic vector $\mathbf{z} \in \mathbb{R}^{d_f}$ which is a distribution over the d_f topics obtained by the topic model such that $\mathbf{z}$ represents the information of the document in terms of the abstract topics. Similarly, each word in the document is represented as a distribution over the set of abstract topics such that topic representations are obtained for the document words as $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_T] \in \mathbb{R}^{T \times d_f}$, where $\mathbf{t}_t \forall\, t \in \{1, \ldots, T\}$ is the topic vector for token x_t. Each of these vectors represent the confidence of the token or document belonging to each of the topics created by the topic model. Therefore, the topic representations provide a "global" topic representation of each word and document by considering the context of the topics found in the corpus as a whole.

6.3.2 Convolutional Layer

The feature vectors obtained by the language and topic models are passed through separate convolutional layers with identical architectures to extract patterns from the respective sequences.

We use a convolutional layer which comprises P convolutional channels with different filter sizes, where each channel groups together different length sequences of word and document embeddings to extract various patterns from the feature representations of the document. Furthermore, we use Q convolutional filters for each convolutional channel, such that each filter learns different patterns from the input sequence to produce Q feature vectors. We add padding (zero vectors) to the start and end of the token sequence, such that all of the convolutional filters step over the token sequence to obtain a T-dimensional output vector $\mathbf{q}_i \forall\, i \in \{1, \ldots, Q\}$. Each convolutional channel combines the outputs from its convolutional filters to obtain a feature matrix

$\mathbf{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_Q] \in \mathbb{R}^{Q \times T}$. We concatenate the features obtained by each convolutional channel to form the output of the convolutional layer.

We prepend the final hidden state of the `[CLS]` token to the final hidden states of the token sequence ($\mathbf{H}$) and pass it to the convolutional layer associated with the PLM to obtain the output as $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_S] \in \mathbb{R}^{S \times T}$ where $S = P \cdot Q$. Similarly, we prepend the document topic vector ($\mathbf{z}$) to the sequence of token topic vectors ($\mathbf{T}$) and pass it through its associated convolutional layer to obtain the output as $\mathbf{R} = [\mathbf{r}_1, \ldots, \mathbf{r}_S] \in \mathbb{R}^{S \times T}$. Finally, we concatenate the output from the two separate convolutional layers to form $\mathbf{G} = [\mathbf{u}_1, \ldots, \mathbf{u}_S, \mathbf{r}_1, \ldots, \mathbf{r}_S] \in \mathbb{R}^{2S \times T}$.

6.3.3 Attention and Output Layers

For our standard architecture shown in Figure 6.1 we consider the label-wise attention mechanism from Vu *et al.* [86] (LAAT) to convert the concatenated feature vectors from the convolutional layers ($\mathbf{G}$) to a matrix $\mathbf{C} \in \mathbb{R}^{L \times T}$, where L is the number of classes and each row $\mathbf{c}_j \forall\, j \in \{1, \ldots, L\}$ is a label-specific vector representation of the text document. As in Section 5.2.2, we refer to this attention mechanism as the general attention (GA) approach which calculates an attention weight matrix $\boldsymbol{\alpha} \in \mathbb{R}^{L \times 2S}$ to represent the attention weights of each feature for each class as follows:

$$\mathbf{Z} = \tanh(\mathbf{Q}_{\mathrm{GA}} \mathbf{G}^T) \tag{6.3.1}$$

$$\boldsymbol{\alpha} = \mathrm{softmax}(\mathbf{U}_{\mathrm{GA}} \mathbf{Z}) \tag{6.3.2}$$

where $\mathbf{Q}_{\mathrm{GA}} \in \mathbb{R}^{d_p \times T}$ and $\mathbf{U}_{\mathrm{GA}} \in \mathbb{R}^{L \times d_p}$ are weight matrices that are learnt during training and d_p is a chosen hyperparameter.

The attention weight matrix is used to calculate the label-wise representations of the document as follows:

$$\mathbf{C} = \boldsymbol{\alpha} \mathbf{G} \tag{6.3.3}$$

where the j-th row ($\mathbf{c}_j$) of the matrix $\mathbf{C} \in \mathbb{R}^{L \times T}$ represents the information of the document with regard to class j.

We also consider the global hierarchical label-wise attention (GHLA) mechanism as proposed in Section 5.2.4 and use the same output layer architecture as discussed in Section 5.2.3. We refer to this architecture as Topic Attention CNN (TopAttCNN) such that TopAttCNN$_{\mathrm{GA}}$ and TopAttCNN$_{\mathrm{GHLA}}$ are the architectures which use the GA and GHLA mechanisms respectively.

6.4 Experiments

6.4.1 Implementation Details

We used the `bert-base-uncased` model from Hugging Face as the PLM in all of our experiments which is the standard BERT model used in previous approaches. We used the Adam optimiser [36] and performed hyperparameter tuning on the learning rate and batch size with possible values of $\{1\text{e-}4, 5\text{e-}4, 1\text{e-}3\}$ and $\{16, 32\}$ respectively. Furthermore, we chose $d_p = 256$ and d_f as the number of classes in the associated dataset as shown in Table 6.1.

Table 6.1: Number of classes in benchmark datasets (d_f).

Dataset	d_f
WOS	141
RCV1-V2	103
NYT	166

We performed experiments on the approach described in Section 6.3 for the two attention mechanisms (TopAttCNN$_{\text{GA}}$ and TopAttCNN$_{\text{GHLA}}$) and use a GA layer for each level of the GHLA mechanism. Furthermore, as a baseline, we evaluate the approach without using the topic model information, i.e., only using the embeddings from the PLM with its associated convolutional layer, to observe the impact that the topic embeddings have on performance. We investigated both of the attention mechanisms for this setting and refer to these approaches as AttCNN$_{\text{GA}}$ and AttCNN$_{\text{GHLA}}$.

We trained the models on the training set for a maximum of 20 epochs and stopped training when the harmonic mean of the Micro-F1 and Macro-F1 on the development set did not increase for 5 consecutive epochs. We chose the models and hyperparameter combination which obtained the highest harmonic mean of Micro-F1 and Macro-F1 on the development set to evaluate the performance of our model on the test set. Our results are reported as the average scores over three runs with different random seeds.

6.4.2 Main Results

Table 6.2 presents the results of the proposed approaches compared to the recent HTC approaches. The results show that the addition of the topic model information generally decreases the performance of the model. The AttCNN$_{\text{GHLA}}$ approach achieves the highest Macro-F1 score on each of the datasets along with the highest Micro-F1 on WOS (85.00) and comparable performance to the best-performing approaches for Micro-F1 on RCV1-V2 (84.54) and NYT (76.94). AttCNN$_{\text{GHLA}}$ only uses the extracted features from the PLM to classify the text documents, showing that the addition of global topic information does not improve classification performance in general. We hypothesise that the topic information of the tokens in the text sequence are often not useful for distinguishing between the different classes since the extracted topics may not correlate with the classes for the particular dataset.

The results show that the GHLA mechanism generally outperforms GA when removing the topic information (AttCNN) whereas TopAttCNN$_{\text{GA}}$ outperforms TopAttCNN$_{\text{GHLA}}$ on all performance measures apart from the Micro-F1 score on NYT. Furthermore, we show that the difference in performance between our four approaches is very small, with most of the metrics having less than a 1% difference. The only metric with a notably large difference is the Macro-F1 score for the TopAttCNN$_{\text{GHLA}}$ model on the RCV1-V2 dataset, which is significantly lower (2.79) than the best-performing approach.

Finally, we observe that the recently proposed HTC approaches outperform the approaches proposed in this chapter on each of the datasets, with large margins for Macro-F1 on the RCV1-V2 and NYT datasets. Notably, the recently proposed approaches use different techniques to fine-tune PLMs whereas the approaches proposed in this chapter use the PLM as a feature extractor.

Table 6.2: Performance comparisons of the AttCNN and TopAttCNN approaches using the three commonly used benchmark datasets.

Model	WOS		RCV1-V2		NYT	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
HiMatch [10]	86.20	80.53	84.73	64.11	–	–
HGCLR [89]	87.11	81.20	86.49	68.31	78.86	67.96
PAAMHiA-T5[1] [31]	**90.36**	81.64	87.22	70.02	77.52	65.97
HBGL [32]	**87.36**	**82.00**	87.23	**71.07**	**80.47**	70.19
HPT [90]	87.16	81.93	**87.26**	69.53	80.42	**70.42**
AttCNN$_{\text{GA}}$	84.93	78.57	84.67	62.48	77.07	64.08
TopAttCNN$_{\text{GA}}$	84.76	78.07	**84.72**	62.33	76.88	64.18
AttCNN$_{\text{GHLA}}$	**85.00**	**79.02**	84.54	**63.11**	76.94	**64.57**
TopAttCNN$_{\text{GHLA}}$	84.64	77.86	84.51	60.32	**77.08**	64.35

[1]Results obtained using twice the number of model parameters as the other approaches.

6.4.3 Stability Analysis

Table 6.3 presents the mean scores and standard deviations over three runs with different random seeds for our proposed approaches. The results show no clear best-performing approach in terms of stability over multiple runs, with most standard deviations being very similar across the three datasets.

However, the TopAttCNN$_{\text{GHLA}}$ obtained a significantly higher standard deviation on the RCV1-V2 Macro-F1 (1.34), which shows that this approach may produce inconsistent results for datasets with a high class imbalance such as RCV1-V2.

Table 6.3: The average performance results of evaluating the AttCNN and TopAttCNN approaches over three independent runs. The values in parentheses show the corresponding standard deviations.

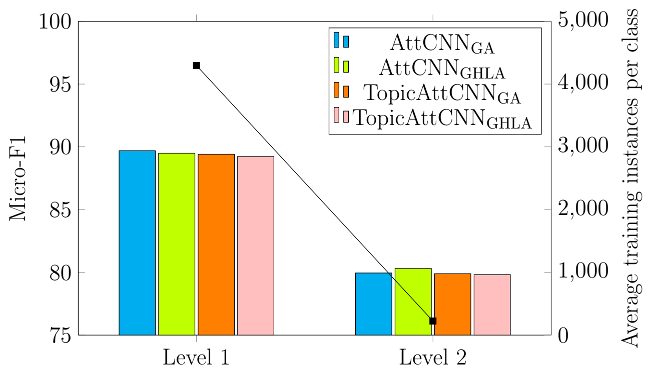
Model	WOS		RCV1-V2		NYT	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
AttCNN$_{\text{GA}}$	84.93 (0.23)	78.57 (**0.25**)	84.67 (0.11)	62.48 (0.29)	77.07 (0.28)	64.08 (0.35)
TopAttCNN$_{\text{GA}}$	84.76 (0.18)	78.07 (0.46)	84.72 (**0.10**)	62.33 (0.28)	76.88 (**0.22**)	64.18 (0.36)
AttCNN$_{\text{GHLA}}$	85.00 (0.27)	79.02 (0.34)	84.54 (0.13)	63.11 (**0.05**)	76.94 (0.42)	64.57 (0.49)
TopAttCNN$_{\text{GHLA}}$	84.64 (**0.17**)	77.86 (0.37)	84.51 (0.11)	60.32 (1.34)	77.08 (0.33)	64.35 (**0.25**)

6.4.4 Level-wise Results

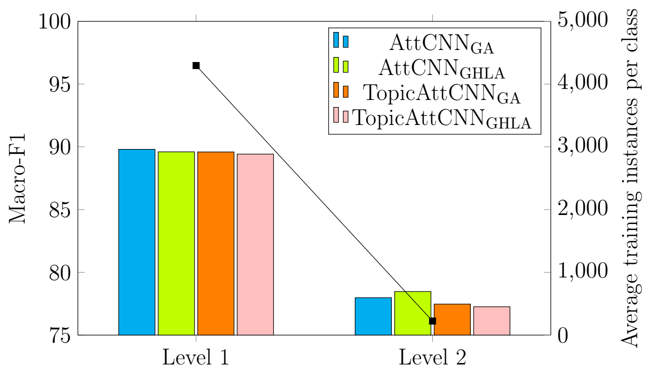
Figure 6.2 presents the Micro-F1 and Macro-F1 scores for the classes at each level for the three benchmark datasets. The level-wise results show the general trend that a higher average number of training instances per class at a certain level leads to improved performance at that level. On the WOS dataset we see that the Micro-F1 and Macro-F1 scores are significantly higher for the first-level classes than the second-level classes where the average number of training instances are much larger for level 1 classes (4295) than level 2 (224). The performance measures on the RCV1-V2 and NYT dataset also show how the average number of training instances per level generally correlates with the performance differences for the classes at each level. However, the level 8 performance on the NYT dataset shows the worst performance even though it has a higher number of training instances per class than level 3 to 7.

6.4.5 Low-resource Results

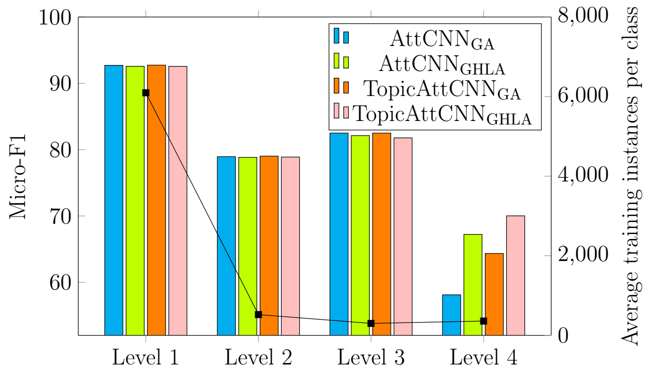
Table 6.4 shows the performance results of the approaches under the low-resource scenario where the training data only consists of 10% of the training set. Performance measures are computed as the average over three runs using different random seeds. The results show that the AttCNN$_{\text{GHLA}}$ and TopAttCNN$_{\text{GA}}$ approaches perform the best on the WOS and NYT datasets respectively. These approaches also obtain the highest Macro-F1 and Micro-F1 on the RCV1-V2 dataset respectively. However, the TopAttCNN$_{\text{GA}}$ perform significantly worse than AttCNN$_{\text{GHLA}}$ on the WOS dataset while the AttCNN$_{\text{GHLA}}$ approach perform similarly to TopAttCNN$_{\text{GA}}$ on the NYT dataset. Therefore, the results show that AttCNN$_{\text{GHLA}}$ generally performs the best in



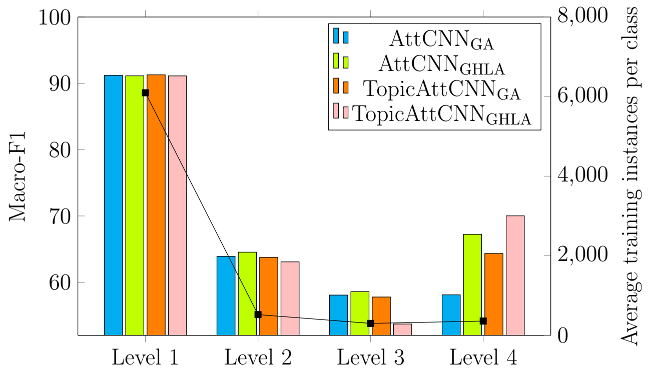
(a) WOS Micro-F1.



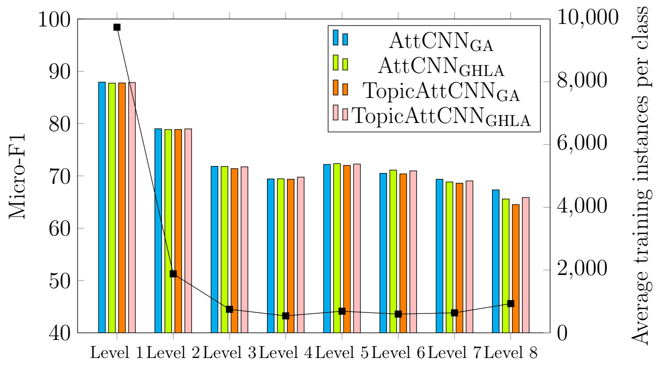
(b) WOS Macro-F1.



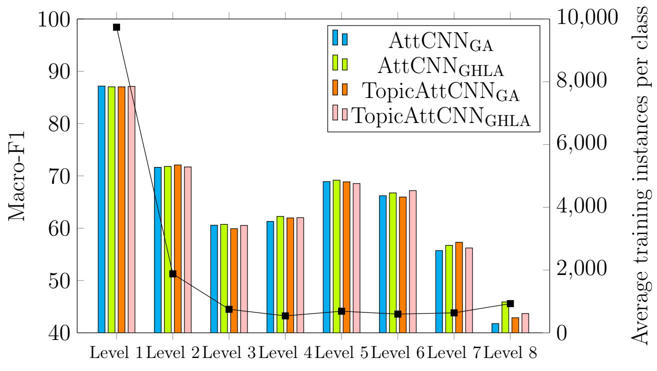
(c) RCV1-V2 Micro-F1.



(d) RCV1-V2 Macro-F1.



(e) NYT Micro-F1.



(f) NYT Macro-F1.

Figure 6.2: Level-wise performance results of the AttCNN and TopAttCNN approaches on the three benchmark datasets. The bar plot gives the F1 scores (left y-axis) for the different approaches at each level of the hierarchy while the line plot shows the average training instances for the classes at a particular level (right y-axis).

the low-resource scenario while AttCNN$_{GA}$ performs slightly worse across the three datasets. Furthermore, the results show that the Macro-F1 scores have a much larger decrease than the Micro-F1 scores when moving from using all the training data to the low-resource scenario. This indicates that the Macro-F1 score is a stricter metric since it captures the per-class performance where as the Micro-F1 hides performance information by biasing the results to the performance of the classes with many instances.

Table 6.4: Performance results of the AttCNN and TopAttCNN approaches under a low-resource scenario where only 10% of the training data is used. For comparison purposes the achieved results when all training data is used are shown in parentheses.

Model	WOS		RCV1-V2		NYT	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
AttCNN$_{\text{GA}}$	75.30 (84.93)	61.76 (78.57)	78.20 (84.67)	46.75 (62.48)	70.80 (77.07)	50.58 (64.08)
AttCNN$_{\text{GHLA}}$	**75.49** (85.00)	**63.89** (79.02)	78.32 (84.54)	**47.44** (63.11)	71.04 (76.94)	50.88 (64.57)
TopAttCNN$_{\text{GA}}$	72.37 (84.76)	57.52 (78.07)	**78.69** (84.72)	46.71 (62.33)	**71.18** (76.88)	**51.34** (64.18)
TopAttCNN$_{\text{GHLA}}$	73.47 (84.64)	59.99 (77.86)	78.35 (84.51)	45.87 (60.32)	70.58 (77.08)	50.96 (64.35)

6.5 Conclusion

In this chapter we proposed a new HTC approach which uses a PLM and topic model to extract features from text documents to train a classifier model. The rationale behind this approach is that the local semantic and global topic representations, obtained by the PLM and topic model respectively, may provide different granularities of the text document and therefore allow the classifier to better distinguish between different classes. The features obtained from the two models are passed through separate convolutional layers after which the outputs are combined and used in label-wise attention and classification layers to obtain the final class predictions. We evaluated this approach with two different label-wise attention mechanisms and compared their performances to a model for which the topic model features were removed. Through comprehensive experiments on three benchmark datasets we showed that the inclusion of the topic model features generally leads to worse performance with the proposed classifier architecture. Finally, we showed that using a PLM as a feature extraction model to train a CNN classifier with label-wise attention mechanisms performs significantly worse than other HTC approaches which fine-tune the parameters of the PLM.

Chapter 7

Introducing Three New Benchmark Datasets for Hierarchical Text Classification

7.1 Introduction

The three datasets used in this thesis thus far (WOS, RCV1-V2, and NYT) are established benchmark datasets for evaluating HTC approaches. However, only the creators of the RCV1-V2 dataset provide a detailed methodology for the creation of their dataset [46]. In particular, the WOS dataset, which is the only benchmark HTC dataset in the domain of research publications, does not provide sufficient detail on how the dataset was created as described in Section 2.10.

We believe it is important to provide a more detailed description of the dataset creation methodology to facilitate reproducibility and reliable comparisons between different classification approaches. Furthermore, detailed dataset creation methodologies enable a better analysis of the results obtained by classification approaches since the characteristics of the dataset may influence the performance of different approaches.

In this chapter, we propose three new datasets for HTC tasks in the domain of research publications. As a starting point for developing the new datasets, we use data that comprises the title and abstract of academic publications from the Web of Science publication database. Our three datasets each use this data but have different classification schemas which determine the categories assigned to the publications.

First, we use a journal-based classification schema from Web of Science which assigns categories to each journal and classifies a publication based on the journal it is published in. However, journal-based classifications have been shown to be unreliable and often inaccurate, with Shu *et al.* [77] showing that these classification schemas may incorrectly classify almost half of publications in some cases. Wang and Waltman [88] compare the two most popular journal-based classification schemas (the WOS subject categories and the Scopus subject areas) and show that they often assign the same categories to publications which do not have strong citation-relationships between them. This indicates that the classifications are too lenient. Furthermore, journal-based classification schemas are not well suited to multidisciplinary journals such as Nature, Science, and PNAS for the obvious reason that these journals span multiple distinct disciplines which leads to publications being incorrectly assigned to all of the categories in the journal. Moreover, Wang and Waltman [88] state that the popularisation of open access multidisciplinary journals such as PLoS ONE and Scientific Reports further increase the unreliability of journal-based classifications.

Our second dataset uses a citation-based classification schema recently developed by Clarivate. Citation-based classification schemas use the citation relationships from a collection of research publications to form clusters of documents which share the same class. However, the citation-based classification schema proposed by Clarivate does not allow a document to belong to multiple research fields, which prevents multi-disciplinary research publications from being appropriately classified.

Due to the shortcomings of these two classification schemas, we also propose an approach that combines these two schemas to create a new categorisation that leverages their respective advantages. In our approach, we filter out categories and documents which do not have a clear overlap between the journal-and citation-based classifications. Therefore, we create new class assignments which are formed by removing assignments and categories that do not form obvious mappings between the two existing classification schemas. The objective of this approach is to increase the probability that an individual document is correctly classified since its classifications are assigned based on its content (journal) as well as its position in the citation network. Furthermore, our proposed approach allows documents to belong to multiple classes, which is important for multi-disciplinary publications, while leveraging the finer-grained citation-based classifications to improve the category assignments of documents.

In summary, we create three new datasets of which the first two use existing journal-and citation-based classifications respectively, while the third dataset uses the classifications obtained by combining these two classifications and applying our proposed filtering approach. Our datasets are unique among benchmark HTC datasets since we sample documents equally for each of the classes in the second level of the hierarchy. Therefore, our datasets are significantly more balanced than the benchmark HTC datasets (see Section 2.10 for the class imbalances of the benchmark datasets).

To evaluate the quality of the three datasets we analyse the semantic similarity between the documents belonging to the different classes. This is done by encoding the documents for each class into a semantic vector embedding and measuring the distances between the documents in the embedding space. We show that the new dataset based on the combination of the journal- and citation-based classification improves the average similarity of the documents belonging to a specific class as well as the separation between documents in different classes.

In alignment with the focus of this thesis, we perform classification experiments on the three proposed datasets with our best-performing proposed approaches: HPTD-ELECTRA, HPTD-DeBERTaV3, and GHLA$_{\text{RoBERTa}}$. These experiments provide further insights to the capabilities of our proposed classification approaches as well as the difficulty in accurately classifying the publications in the three proposed datasets. Through the experiments we show that the HPTD-DeBERTaV3 and GHLA$_{\text{RoBERTa}}$ models generally obtain the best performances across the three datasets.

7.2 Background

7.2.1 Web of Science Subject Categories

The WOS subject categories form a classification schema which categorises research publications based on the journal, conference, or book in which they are published. For sake of brevity, we refer to all research publication venues as journals for the remainder of this thesis.

In the WOS subject categories classification schema, each journal is assigned to one or more categories from 256 possible classes which cover all thematic areas of scientific research. Therefore, a document is assigned to the classes associated with the journal it is published in. The WOS subject categories are not structured hierarchically, i.e., they constitute a multi-label classification framework. Table 7.1 shows the title and abstract of example publications with their assigned WOS categories.

The detailed methodology for assigning WOS categories to a journal is not published, but these classifications consider many aspects which include: the content and scope of the journal, affiliations of authors and editors of the journal, citation relationships of publications in the journal [64], funding agencies and sponsors, as well as the journal's categorisation in other bibliographic databases.

7.2.2 CREST Journal-based Classification Schema

The Centre for Research on Evaluation, Science and Technology (CREST) is a research centre in the Faculty of Arts and Social Sciences at Stellenbosch University which focuses on various topics including bibliometrics, scientometrics, and research evaluation. CREST devised a new classification schema with

Table 7.1: Example publications with associated WOS subject categories. The "Publication" column comprises the title and truncated abstract of the publication.

Publication	WOS categories
"Can Creditor Bail-in Trigger Contagion? The Experience of an Emerging Market. The successful bail-in of creditors in African Bank, a small South African monoline lender, provides an opportunity to evaluate the intended and unintended consequences of new resolution tools. Using a dataset that matches quarterly, daily, and financial-instrument level data, I show that the bail-in led to money-market funds 'breaking the buck,' triggering significant redemptions and some financial contagion..."	Business Finance Economics
"Dissecting the genre of Nigerian music with machine learning models. Music Information Retrieval (MIR) is the task of extracting high-level information, such as genre, artist or instrumentation from music. Genre classification is an important and rapidly evolving research area of MIR. To date, only a small amount of research work has been done on the automatic genre classification of Nigerian songs. Hence, this study presents a new music dataset, namely the ORIN dataset, consisting of only Nigerian songs..."	Computer Science Information Systems
"The complementarity of a diverse range of deep learning features extracted from video content for video recommendation. Following the popularisation of media streaming, a number of video streaming services are continuously buying new video content to mine the potential profit from them. As such, the newly added content has to be handled well to be recommended to suitable users. In this paper, we address the new item cold-start problem by exploring..."	Computer Science Artificial Intelligence Engineering (Electrical & Electronic) Operations Research & Management Science

a two-level hierarchical class structure as an alternative to the WOS subject categories.

They mapped the WOS categories to a two-level hierarchical class structure with fewer classes to balance the size of class clusters and cover strategic fields of research. Their schema comprises 7 and 55 classes for the first and second levels of the class hierarchy respectively. Table 7.2 provides examples of the mappings from WOS categories to the classification schema proposed by CREST which we refer to as Journal-based Topics (JT), where JT_{L1} and JT_{L2} represent the classes for level 1 and level 2 respectively. Table 7.3 shows example publications with their JT classifications.

Table 7.2: Examples of mapping from WOS categories to JT.

WOS	JT_{L1}	JT_{L2}
Agronomy	Agricultural sciences	Agronomy
Forestry	Agricultural sciences	Agricultural sciences (Other)
Literature	Humanities and arts	Language & linguistics
Art	Humanities and arts	Other humanities & arts
Geology	Natural sciences	Geosciences
Behavioral Sciences	Social sciences	Psychology

Table 7.3: Example publications with associated JT classifications. The "Publication" column comprises the title and truncated abstract of the publication.

Publication	JT_{L1}	JT_{L2}
"Can Creditor Bail-in Trigger Contagion? The Experience of an Emerging Market..."	Social Sciences	Business Economics
"Dissecting the genre of Nigerian music with machine learning models. Music Information..."	Natural sciences	Information, computer & communication technologies
"The complementarity of a diverse range of deep learning features extracted from video content for video recommendation. Following the popularisation of media streaming, a number of video streaming services are..."	Engineering Natural sciences	Electrical & electronic engineering Engineering sciences (other) Information, computer & communication technologies

7.2.3 Clarivate Citation Topics Classification Schema

Clarivate proposed the Citation Topics (CT) classification schema, which clusters documents based on citation relationships, as an alternative to journal-based classifications such as the WOS subject categories.

Although the detailed methodology for the classification algorithm has not been published, the classifications are obtained by a Leiden community detection algorithm [83] based on the citation relationships between documents. Community detection algorithms have the objective of identifying communities (or clusters) of nodes within a network that are more densely connected to each other than to nodes outside of the community [83]. These algorithms typically use an objective function which measures the quality of the node partitions with the aim of maximising the intra-community connections while minimising the inter-community connections. The Leiden community detection algorithm proposes several improvements over the popular Louvain community detection algorithm [8] to obtain a better partition of nodes in a network.

The Louvain algorithm starts with each node forming its own community and attempts to optimise the modularity of the network which is calculated as:

$$\mathcal{M} = \frac{1}{2m}\sum_{i=c}^{C}(e_c - \gamma\frac{K_c^2}{2m}) \tag{7.2.1}$$

where m is the total number of edges in the network, C is the number of communities, and K_c is the sum of the degrees of the nodes in community c such that $\frac{K_c^2}{2m}$ represents the expected number of edges in the community. Furthermore γ is a resolution parameter which determines the number of communities formed by the algorithm since higher γ values lead to more communities being formed. Therefore, the objective of the algorithm is to maximise the difference between the expected and true number of edges in a community. Note that the Louvain algorithm can be optimised through other objective functions [83].

The Louvain algorithm uses two phases to assign nodes to suitable communities. In the first phase, each node is considered to be moved to neighbouring communities and the change in objective function is determined. The node movements that maximise the increase in the objective function are performed until no improvement in the objective function can be achieved. In the second phase, the nodes in the communities obtained by the first phase are aggregated to form individual nodes where the edge weights between the nodes are determined by the sum of the weights of the edges between the original nodes in the communities. The first phase is applied to the aggregated nodes and this procedure is repeated until the objective function can no longer be improved. Traag *et al.* [83] show that the Louvain algorithm may obtain arbitrarily badly connected communities and even internally disconnected communities, i.e., communities where a section of the community can only reach another section of the community through a path that goes outside of that community.

The Leiden algorithm addresses these shortcomings by proposing several improvements to the Louvain algorithm. Similar to the Louvain algorithm, each node starts off as its own community and the nodes are moved to different communities to maximise an objective function. However, the Leiden algorithm introduces an approach which refines these communities obtained by the first phase such that each community may be split into multiple subcommunities. The refinement phase uses the partitions obtained by the first phase to locally merge the nodes in each community to potentially form subcommunities. In the refinement phase, nodes are not always greedily assigned to the community which maximises the objective function increase. Alternatively, any node assignment which results in an increase in the objective function is considered. The assignments are selected with a certain probability based on their increase in the objective function such that assignments with larger increases are more likely. This non-greedy merging in the refinement phase enables a better exploration of the partition space [83]. The aggregate network is created based on the refined partitions which increases the probability of finding high-quality communities. Using these improvements, the

Leiden algorithm guarantees that formed communities are well connected and that it converges to a solution where all subsets of communities are guaranteed to be locally optimal.

In the CT classification schema, the document communities (or clusters) obtained by the Leiden algorithm are used to assign documents to a three-level hierarchical class structure. The first and second level classes ($\mathrm{CT}_{\mathrm{L1}}$ and $\mathrm{CT}_{\mathrm{L2}}$) are manually labelled based on the contents of the documents in their clusters, while the third level classes ($\mathrm{CT}_{\mathrm{L3}}$) are labelled algorithmically with the most significant keyword in the cluster. The three levels of the class hierarchy comprise 10, 326, and 2,457 classes respectively. Table 7.4 shows example publications with their CT classifications.

Table 7.4: Example publications with associated CT classifications. The "Publication" column comprises the title and truncated abstract of the publication.

Publication	$\mathrm{CT}_{\mathrm{L1}}$	$\mathrm{CT}_{\mathrm{L2}}$	$\mathrm{CT}_{\mathrm{L3}}$
"Can Creditor Bail-in Trigger Contagion? The Experience of an Emerging..."	Social Sciences	Economics	Economic Growth
"Dissecting the genre of Nigerian music with machine learning models. Music Information..."	Electrical Engineering, Electronics & Computer Science	Knowledge Engineering & Representation	Statistical Tests
"The complementarity of a diverse range of deep learning features extracted from video content for video..."	Electrical Engineering, Electronics & Computer Science	Knowledge Engineering & Representation	Collaborative Filtering

7.3 Methodology

From the WOS publication database, we randomly sampled 5,000 papers for each of the $\mathrm{CT}_{\mathrm{L2}}$ classes, resulting in 1,630,000 records. Each record contains the title and abstract of the publication, along with the JT and CT classifications. We use this dataset to create three HTC datasets which include:

- $\mathrm{WOS}_{\mathrm{JT}}$, which only uses the journal-based JT classifications.
- $\mathrm{WOS}_{\mathrm{CT}}$, which only uses the citation-based CT classifications.
- $\mathrm{WOS}_{\mathrm{JTF}}$, which uses the JT Filtered (JTF) classification schema that filters out documents and classes which do not have a clear overlap between JT and CT classifications.

Table 7.5 shows the summary statistics for the three datasets. We split each dataset into train (70%), development (15%), and test (15%) sets.

Table 7.5: Characteristics of the newly created HTC datasets. The column "Levels" gives the number of levels, while "Classes$_{\mathrm{L1}}$" and "Classes$_{\mathrm{L2}}$" give the number of first- and second-level classes in the class structure. "Avg. Classes" is the average number of classes per document, while "Train", "Dev", and "Test" are the number of instances in each of the dataset splits.

Dataset	Levels	Classes$_{\mathrm{L1}}$	Classes$_{\mathrm{L2}}$	Avg. Classes	Train	Dev	Test
WOS$_{\mathrm{JT}}$	2	6	52	2.93	30,356	6,505	6,505
WOS$_{\mathrm{CT}}$	2	10	326	2.00	45,640	9,780	9,780
WOS$_{\mathrm{JTF}}$	2	6	46	2.25	30,048	6,439	6,439

7.3.1 WOS$_{\mathrm{JT}}$ Dataset

To create the WOS$_{\mathrm{JT}}$ dataset we use the JT classifications as described above but remove the JT classes for which there are no instances in the dataset. We randomly sample 1,000 documents for each JT$_{\mathrm{L2}}$ class with the aim of creating a dataset that is balanced at the second level of the class hierarchy. Since a document can be assigned to one or more JT$_{\mathrm{L2}}$ classes, the second level classes are not perfectly balanced and the dataset contains 43,366 documents in total.

Figure 7.1 presents the first-level classes of the WOS$_{\mathrm{JT}}$ dataset along with the associated number of children classes and the number of documents assigned to each class. This figure shows how the number of documents assigned to a first-level class is larger for those classes with a larger number of children classes due to our sampling strategy.

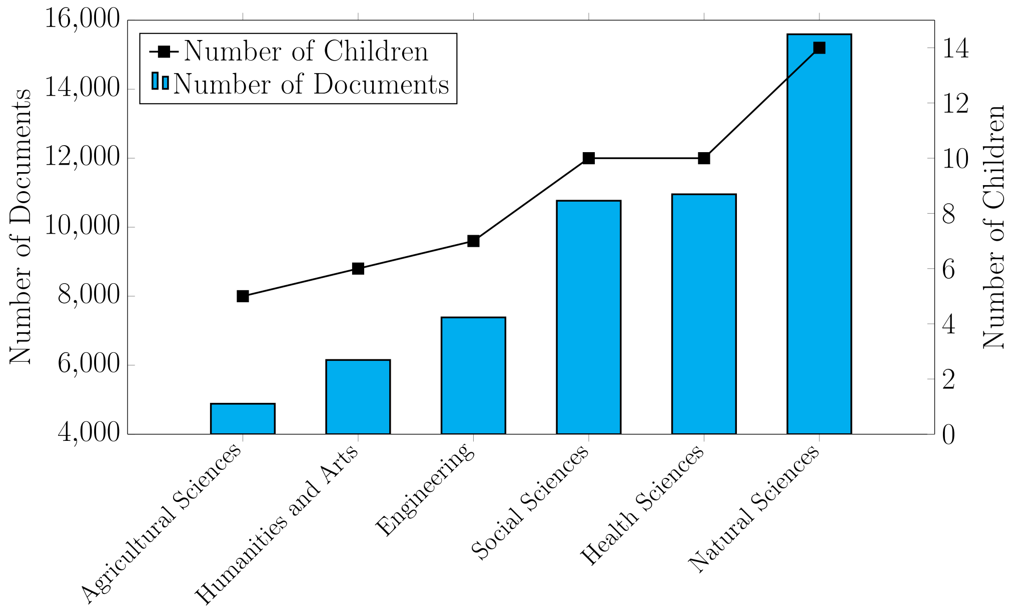
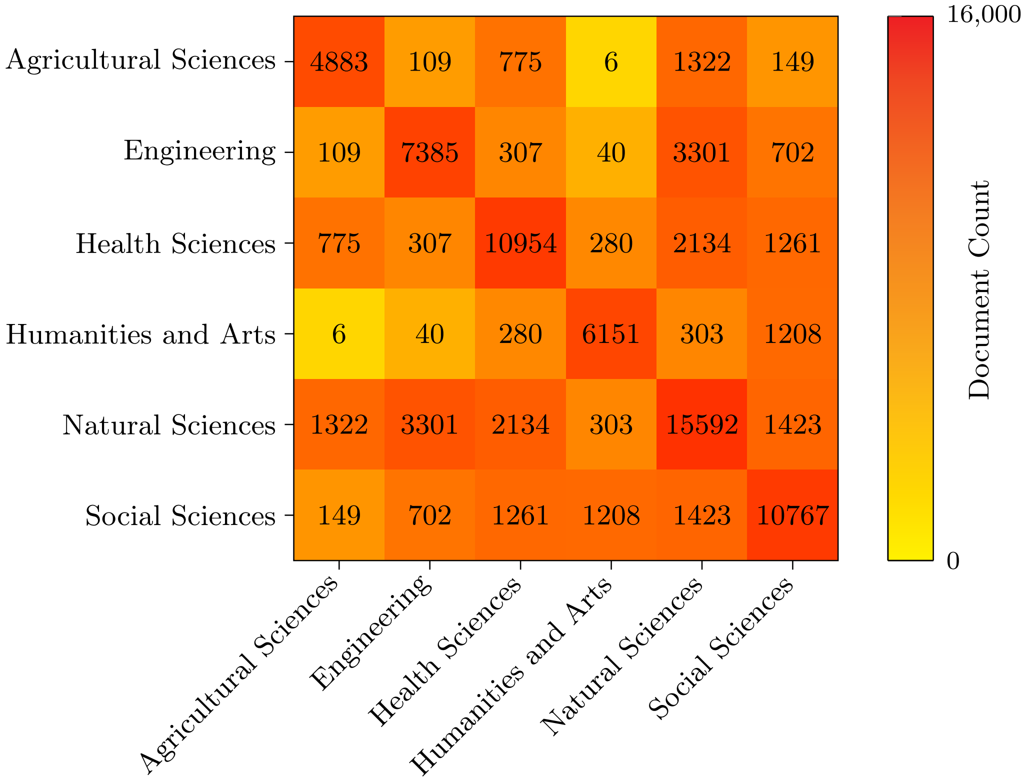


Figure 7.1: Characteristics of first-level classes of the WOS$_{\mathrm{JT}}$ dataset. The bar plot gives the number of documents assigned to each class (left y-axis) while the line plot shows the number of children for each class (right y-axis).

Figure 7.2 presents the co-occurrence document counts for the first-level classes, i.e., the number of times a document with a certain class is also assigned to each of the other classes. This figure shows that categories such as “Natural sciences” and “Engineering” are much more likely to co-occur than other categories, for example, “Humanities and arts” and “Agricultural sciences”.

Figure 7.3 presents the distribution of the number of documents assigned to each of the classes in the second level of the hierarchy. From this figure we can see that most JT_{L2} classes have around 1,000 document assignments. However, the “Ornithology” class has less than 1,000 document assignments since it only has 588 documents available to sample from. Furthermore, the “Other social sciences”, “Clinical and public health (other)”, and “Engineering sciences (other)” classes have 3,553, 3,900, and 3,984 documents respectively since they often overlap with other class assignments.



	Agricultural Sciences	Engineering	Health Sciences	Humanities and Arts	Natural Sciences	Social Sciences
Agricultural Sciences	4883	109	775	6	1322	149
Engineering	109	7385	307	40	3301	702
Health Sciences	775	307	10954	280	2134	1261
Humanities and Arts	6	40	280	6151	303	1208
Natural Sciences	1322	3301	2134	303	15592	1423
Social Sciences	149	702	1261	1208	1423	10767

Figure 7.2: Co-occurrence document counts for first-level classes of the WOS_{JT} dataset.

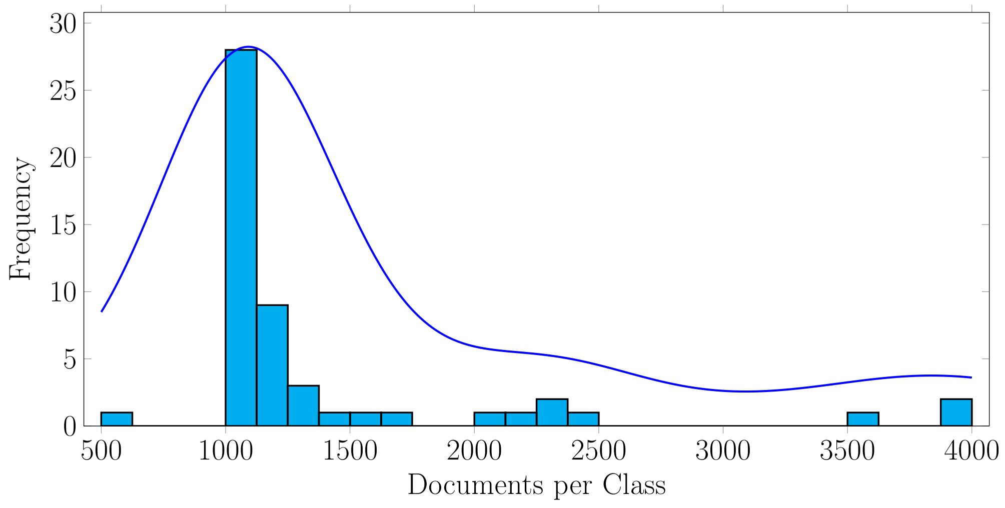


Figure 7.3: Distribution of the number of documents assigned to each of the JT_{L2} classes in the WOS_{JT} dataset.

7.3.2 WOS_{CT} Dataset

We create the WOS_{CT} dataset by only using the citation-based CT classifications as described above. However, we only use the first two levels of the class hierarchy since the third level consists of 2,457 classes which leads to a severely imbalanced dataset with certain classes at the third level having a much higher number of documents than others. Therefore, WOS_{CT} uses 10 first-level (CT_{L1}) and 326 second-level (CT_{L2}) classes. We randomly sample 200 documents for each CT_{L2} class to create the WOS_{CT} dataset with 65,200 academic publications. Since each document is assigned a single class per level, there are no co-occurrences between the classes at the same level, resulting in a perfectly balanced dataset at the second level with exactly 200 documents per CT_{L2} class.

Figure 7.4 shows all of the CT_{L1} classes of the WOS_{CT} dataset along with the corresponding number of children classes and number of documents. It should be noted that the CT_{L1} classes with a larger number of children classes also have more assigned documents. Particularly, the "Clinical & Life Sciences" class has 132 children classes, so it has a much larger number of assigned documents compared to the other classes.

7.3.3 WOS_{JTF} Dataset

The third proposed dataset combines the JT and CT classifications with the goal of assigning classifications that capture the content of the publications more accurately. As shown above, the original dataset contains 52 and 326 JT_{L2} and CT_{L2} classes respectively. Therefore, to combine the journal-based and citation-based classifications we attempt to map each CT_{L2} class to a JT_{L2} class and remove categories and documents which do not have a clear

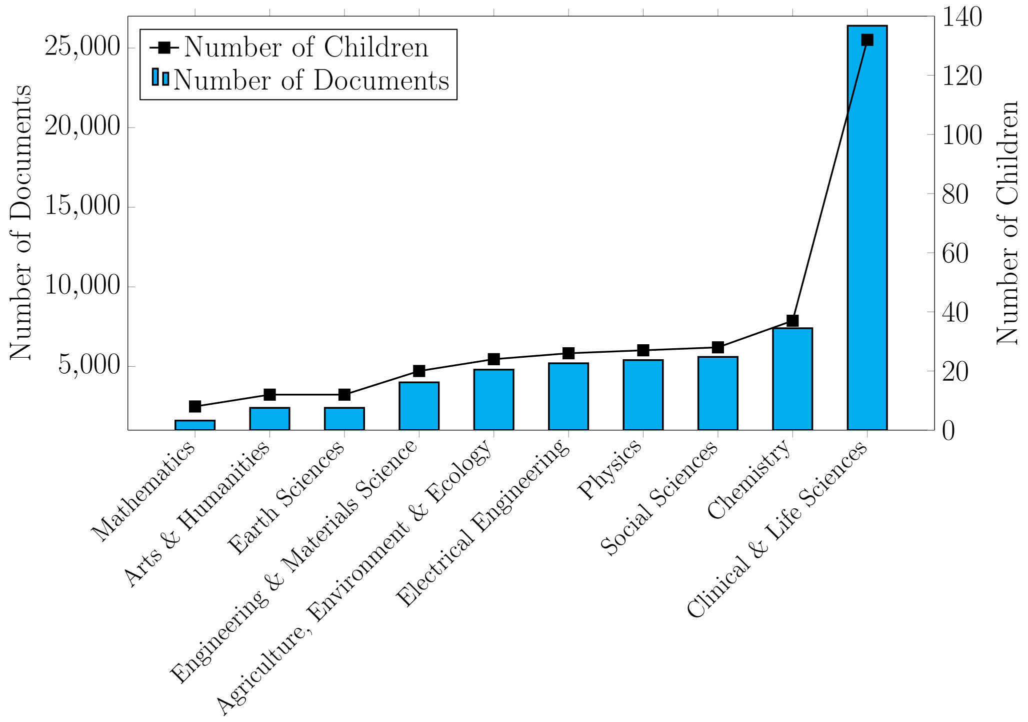


Figure 7.4: Characteristics of first-level classes of the WOS_{CT} dataset. The bar plot gives the number of documents assigned to each class (left y-axis) while the line plot shows the number of children for each class (right y-axis).

mapping between the two classification frameworks. The rationale behind this approach is to leverage the respective advantages of the two classifications to create document assignments that are closely linked in the citation network and are similar based on their content.

We start by obtaining the co-occurrence counts between each CT_{L2} and JT_{L2} class as a matrix $\mathbf{M} \in \mathbb{Z}^{326\times 52}$ where each row $\mathbf{m}_i \forall\, i \in \{1, \ldots, 326\}$ represents the co-occurrence counts between the i-th CT_{L2} class ($\text{CT}_{\text{L2},i}$) and all JT_{L2} classes. In other words, for all the documents that are assigned to $\text{CT}_{\text{L2},i}$, $\mathbf{m}_i$ captures the number of documents assigned to each of the JT_{L2} classes. Therefore, $m_{i,j} \forall\, j \in \{1, \ldots, 52\}$ is the number of times that the j-th JT_{L2} class ($\text{JT}_{\text{L2},j}$) is assigned to a document belonging to $\text{CT}_{\text{L2},i}$.

We use the co-occurrence matrix to find the most relevant JT_{L2} classes for each $\text{CT}_{\text{L2},i}$ class. First, we find the highest co-occurrence count of $\mathbf{m}_i$ as $k_i = \max(\mathbf{m}_i)$ and divide k_i by the co-occurrence count for each JT_{L2} class to obtain a ratio for each JT_{L2} class as:

$$\mathbf{r}_i = \left[\frac{k_i}{m_{i,1}}, \ldots, \frac{k_i}{m_{i,52}}\right] \tag{7.3.1}$$

where $r_{i,j}$ is the j-th element in $\mathbf{r}_i$ that represents the ratio of the highest co-occurrence count (k_i) to $\text{JT}_{\text{L2},j}$. In other words, it represents the number of times that k_i is greater than the count of $\mathrm{JT}_{\mathrm{L2},j}$. We use these ratios to create a set of $\mathrm{JT}_{\mathrm{L2}}$ classes to which $\mathrm{CT}_{\mathrm{L2},i}$ is mapped, controlled by a threshold (γ):

$$Q_i = \{\mathrm{JT}_{2,j} \forall j \in \{1, \ldots, 52\} | r_{i,j} \leq \gamma\} \tag{7.3.2}$$

We choose a threshold of $\gamma = 1.5$, such that a $\mathrm{CT}_{\mathrm{L2}}$ class is only mapped to a $\mathrm{JT}_{\mathrm{L2}}$ class if the highest co-occurrence count for the particular $\mathrm{CT}_{\mathrm{L2}}$ class is less than 1.5 times the co-occurrence count of the two classes. Mappings which have fewer overlapping documents relative to the highest overlapping mapping are removed such that only the most common co-occurrences between the two classification schemas form part of the final mapping.

In order to guarantee that no misassignments occurred, two annotators individually checked each set Q_i and removed $\mathrm{JT}_{\mathrm{L2}}$ classes from the set which are clearly inaccurate mappings from $\mathrm{CT}_{\mathrm{L2},i}$. We calculate the Cohen's kappa score to determine the inter-annotator agreement as:

$$kappa = \frac{P_o - P_e}{1 - P_e} \tag{7.3.3}$$

where P_o is the proportion of items that the annotators agree on and P_e is the expected agreement when annotators assign labels randomly based on empirical priors. The Cohen's kappa score obtained by the two annotators for the removal of mappings was 0.85 which indicates that there was a very strong agreement between the decisions made by the two annotators. After the initial annotation, the two annotators discussed their decision differences and agreed on the final set of mappings to remove. Table 7.6 lists all inaccurate mappings that were removed. We use the filtered Q_i set and remove 24,215 documents that belong to a $\mathrm{CT}_{\mathrm{L2}}$ class with an empty mapping set $Q_i = \emptyset$. Furthermore, we remove all documents that belong to $\mathrm{CT}_{\mathrm{L2},i}$ but have a $\mathrm{JT}_{\mathrm{L2}}$ class that is not in Q_i which results in 825,529 document removals. The reasoning behind this approach is to improve the quality of class assignments by only allowing documents which form part of the clear mapping from $\mathrm{CT}_{\mathrm{L2}}$ to $\mathrm{JT}_{\mathrm{L2}}$ classes to remain in the dataset.

We remove the $\mathrm{JT}_{\mathrm{L2}}$ classes that do not have any suitable $\mathrm{CT}_{\mathrm{L2}}$ classes that map to them based on the removal of mappings from the two annotators as described above. These classes include: "Risk Assessment", "Operations Research & Management Science", "Contamination & Phytoremediation", "Herbicides, Pesticides & Ground Poisoning", "Sports Science", and "Ornithology". We refer to the resulting class set as JT Filtered (JTF) which comprises 6 first-level ($\mathrm{JTF}_{\mathrm{L1}}$) and 46 second-level ($\mathrm{JTF}_{\mathrm{L2}}$) classes.

To create the $\mathrm{WOS}_{\mathrm{JTF}}$ dataset, we sample 1,000 documents for each of the $\mathrm{JTF}_{\mathrm{L2}}$ classes to obtain a total of 42,926 instances. Figure 7.5 presents the first-level classes of the $\mathrm{WOS}_{\mathrm{JTF}}$ dataset with the associated number of children classes and number of documents assigned to each class while Figure 7.6 presents the co-occurrence counts for the first-level classes. From Figure 7.6

Table 7.6: Removed mappings from CT_{L2} to JT_{L2}.

CT_{L2}	JT_{L2}
Soil Science	Other social sciences
Diarrhoeal Diseases	Other social sciences
Water Treatment	Other social sciences
Forestry	Other social sciences
Nuclear Geology	Other social sciences
Bioengineering	Other social sciences
Folklore & Humor	Psychology
Electrical Protection	Other earth sciences
Electrical - Sensors & Monitoring	Materials sciences
Remote Research & Education	Engineering sciences (other)

we can see that there is a much clearer distinction between the first-level class assignments in the WOS_{JTF} dataset compared to the WOS_{JT} dataset, with fewer instances overlapping multiple first-level classes.

Figure 7.7 gives the distribution of documents assigned to the second-level classes of WOS_{JTF}. This figure shows that most classes have around 1,000 document assignments. However, "Biochemistry & molecular biology" and "Marine & freshwater biology" only have 450 and 594 documents respectively since they do not have 1,000 documents to sample from. Furthermore, "Clinical and public health (other)" and "Engineering sciences (other)" have 1,979 and 2,045 documents respectively since they often co-occur with other classes.

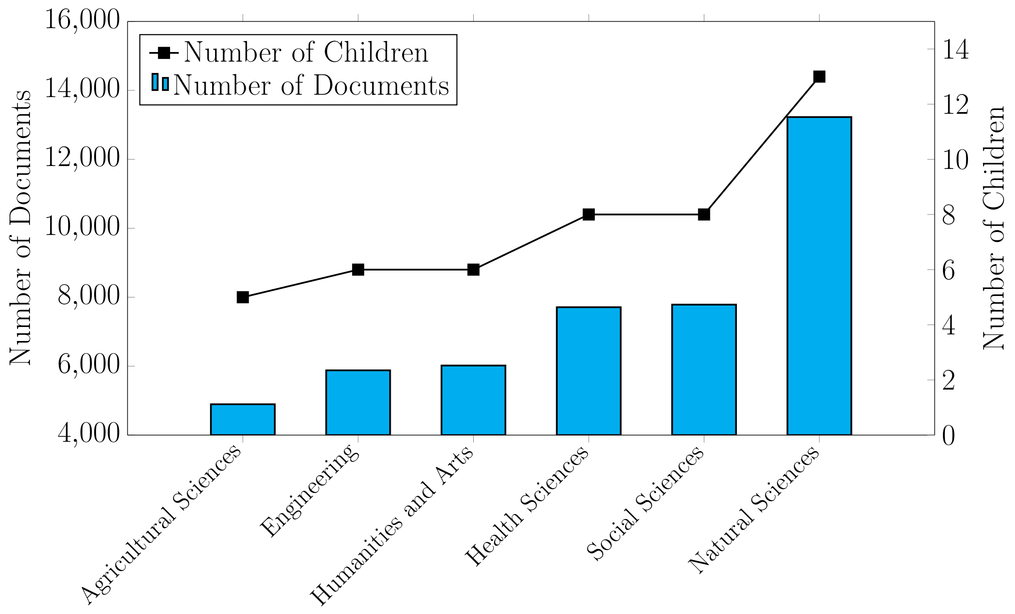


Figure 7.5: Characteristics of first-level classes of the WOS_{JTF} dataset. The bar plot gives the number of documents assigned to each class (left y-axis) while the line plot shows the number of children for each class (right y-axis).

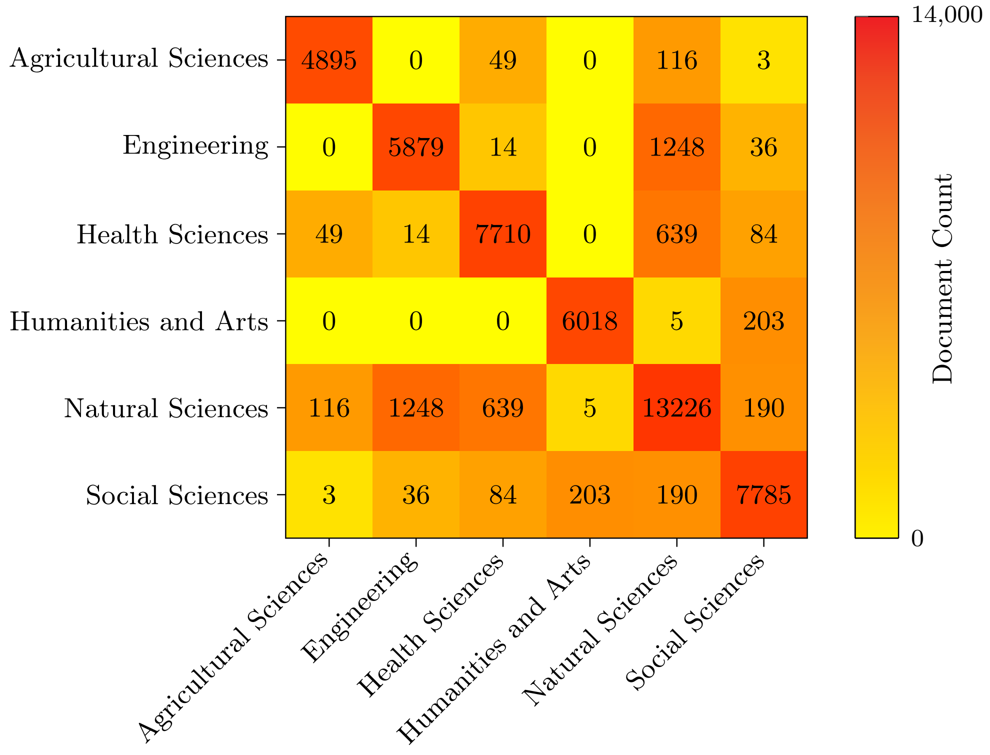


Figure 7.6: Co-occurrence counts for first-level classes of the WOS_{JTF} dataset.

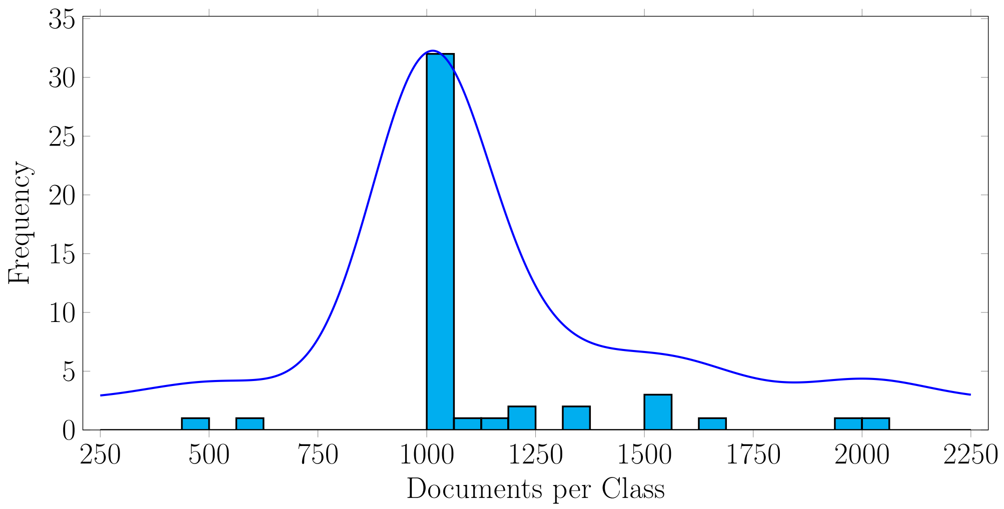


Figure 7.7: Distribution of the number of documents assigned to each of the JTF_{L2} classes in the WOS_{JTF} dataset.

7.4 Experiments

7.4.1 Cluster Analysis

We evaluate the quality of the three datasets by analysing the semantic similarity between the documents belonging to each of the classes (or clusters). We investigate the semantic similarity between the documents within each class-cluster as well as the separation between documents in different clusters to determine whether the class assignments form cohesive clusters of documents that are closely related thematically. We evaluate the class assignments separately for the two levels of the class hierarchy.

First, we use a sentence-BERT model [68] to convert each document in the dataset to a fixed-sized embedding which captures the semantic meaning of the document. A sentence-BERT model is a modified BERT model that is tuned to produce a semantic embedding for a sentence or document in contrast to the standard BERT model which produces embeddings for each token in the input sequence. We use the `all-mpnet-base-v2` sentence-BERT model from the SentenceTransformers library. Furthermore, we cluster (or group) all of the documents based on their first- or second-level class assignments. Documents that belong to more than one class at a level are duplicated and placed into each of the class-clusters they belong to.

Suppose we have L clusters for a particular level of a dataset. We use the document embeddings to calculate the average cosine similarity between the instances in each cluster $j \in \{1, \ldots, L\}$ as:

$$o_j = \frac{1}{N_j(N_j - 1)} \sum_{k=1}^{N_j} \sum_{l=1, k \neq l}^{N_j} \text{CosSim}(\mathbf{x}_k, \mathbf{x}_l) \tag{7.4.1}$$

where N_j is the number of instances that belong to cluster j and CosSim is the cosine similarity function. The cosine similarity function between the embeddings of two documents is calculated as:

$$\text{CosSim}(\mathbf{x}_k, \mathbf{x}_l) = \frac{\mathbf{x}_k \cdot \mathbf{x}_l}{\|\mathbf{x}_k\| \cdot \|\mathbf{x}_l\|} \tag{7.4.2}$$

such that semantically similar documents have high similarity scores.

To determine how well-separated the documents in the different clusters are, we calculate the silhouette score for each instance $\mathbf{x}_i \forall i \in \{1, \ldots, N\}$, where N is the number of instances in the dataset. The silhouette score measures the quality of clusters by quantifying how well-separated and internally cohesive the clusters are. The silhouette score for instance i is calculated as:

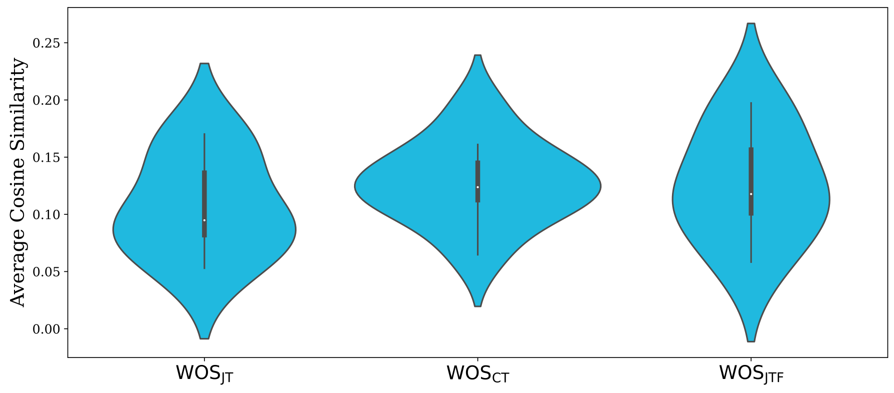
$$s_i = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{7.4.3}$$

where $a(i)$ is the average distance from instance i to all other instances in the same cluster which we calculate with the cosine distance function given by:

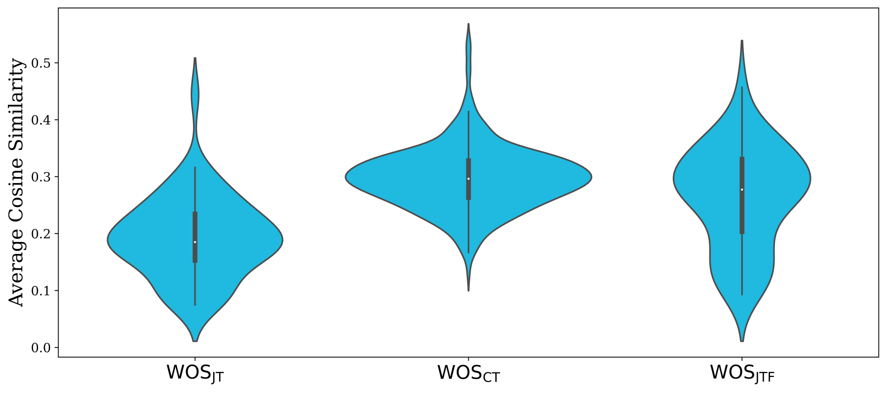
$$\text{CosDist}(\mathbf{x}_k, \mathbf{x}_l) = 1 - \text{CosSim}(\mathbf{x}_k, \mathbf{x}_l) \tag{7.4.4}$$

and $b(i)$ is the average distance from instance i to all instances in the nearest neighbouring cluster. The nearest neighbouring cluster is determined by minimising the average distance from the instance to the instances of a different cluster. The silhouette score ranges from -1 to 1 where a high silhouette score (1) indicates that instances are semantically similar within a cluster and are far apart from other clusters while a low silhouette score (-1) indicates that instances are not well-clustered and the clusters have a high overlap.

The violin plots in Figure 7.8 gives the cosine similarity distributions over the classes in the first and second level for the three datasets. These plots show that the semantic similarity between documents in the same classes is generally higher in the WOS_{JTF} dataset than the WOS_{JT} dataset. This shows



(a) First-level classes.

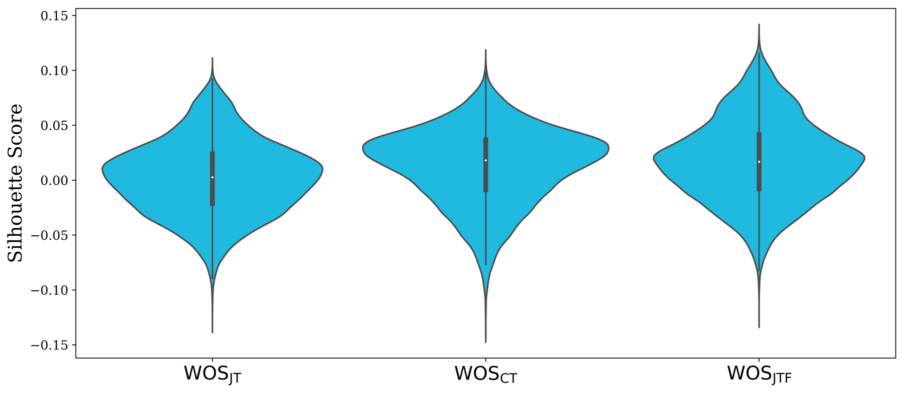


(b) Second-level classes.

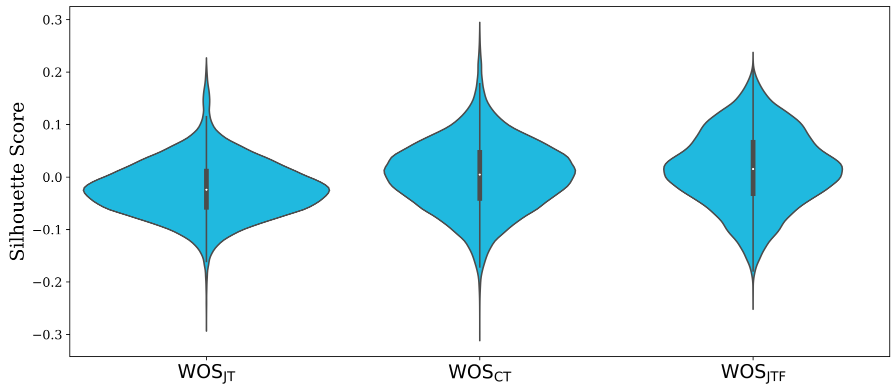
Figure 7.8: Violin plots of the cosine similarity per class-cluster for the three newly proposed datasets.

that the proposed filtering approach was able to leverage the citation-based classifications to improve the classification of the documents such that the documents belonging to a certain class are on average more semantically similar than the $\mathrm{WOS}_{\mathrm{JT}}$ assignments.

Figure 7.9 presents the violin plots for the silhouette scores over all of the instances for the first and second level classes on the three datasets. The figures show that the $\mathrm{WOS}_{\mathrm{JTF}}$ dataset generally has a higher silhouette score than the $\mathrm{WOS}_{\mathrm{JT}}$ dataset, especially for the second level of the class hierarchy. This indicates that our approach to create the $\mathrm{WOS}_{\mathrm{JTF}}$ dataset was able to effectively remove instances and categories based on the citation-based classifications in order to improve the separation in terms of semantic similarity between the class-clusters on average. However, the silhouette scores are still very low across the three datasets which implies that many of the clusters are not well-separated and may overlap.



(a) First-level classes.



(b) Second-level classes.

Figure 7.9: Violin plots of the silhouette scores for the three newly proposed datasets.

7.4.2 Classification Results

We perform experiments on the three newly created datasets with the three best-performing approaches proposed in this thesis: HPTD-ELECTRA, HPTD-DeBERTaV3, and $\text{GHLA}_{\text{RoBERTa}}$. For each approach we use the same architecture and hyperparameter tuning procedure as mentioned in their associated chapters. Furthermore, we evaluate the performance of the HPT approach on the three datasets using the same hyperparameters and tuning procedure as the original paper.

Table 7.7 presents the results of the different approaches on the three newly created datasets. The results show that $\text{GHLA}_{\text{RoBERTa}}$ and HPTD-DeBERTaV3 generally outperform the other approaches and they obtain very similar results for each of the datasets. The largest performance difference between $\text{GHLA}_{\text{RoBERTa}}$ and HPTD-DeBERTaV3 is an improvement of 0.19 for the WOS_{JT} Macro-F1 score, showing that they perform consistently similar on each of the datasets. Furthermore, the HPT approach obtains the highest Macro-F1 score on the WOS_{CT} dataset and outperforms HPTD-ELECTRA on all three datasets.

The results show that the classification performance is significantly improved on the WOS_{JTF} dataset compared to WOS_{JT}. For example, the Micro-F1 and Macro-F1 scores of the $\text{GHLA}_{\text{RoBERTa}}$ approach increase by 17.34 and 20.54 respectively when moving from WOS_{JT} to WOS_{JTF}. We believe that this increase in performance is due to the filtering approaches used to remove documents and categories based on the mapping from the CT classifications. This indicates that the WOS_{JTF} dataset contains more accurate classifications (or fewer incorrectly assigned documents) which can be more effectively learnt by the classification models. However, other factors such as the number of classes and the average classes per instance may also explain a proportion of the observed performance differences.

Table 7.7: Performance comparisons of the best-performing approaches on the three newly proposed datasets.

Model	WOS_{JTF}		WOS_{JT}		WOS_{CT}	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
HPT	84.97	82.13	67.62	61.71	73.25	**61.87**
HPTD-ELECTRA	84.75	81.70	67.19	60.91	71.39	58.41
HPTD-DeBERTaV3	85.68	**82.93**	68.35	62.19	**73.45**	61.27
$\text{GHLA}_{\text{RoBERTa}}$	**85.72**	82.92	**68.38**	**62.38**	73.34	61.29

Table 7.8 presents the results of the different approaches with standard deviations over three runs with different seeds. The $\text{GHLA}_{\text{RoBERTa}}$ and HPT approaches generally have the most stable results over multiple runs with $\text{GHLA}_{\text{RoBERTa}}$ obtaining the lowest standard deviations on the WOS_{JTF} and

$\mathrm{WOS_{JT}}$ datasets and HPT having the lowest standard deviation on the $\mathrm{WOS_{CT}}$ dataset.

Table 7.8: The average performance results of evaluating the models over three independent runs on the newly proposed datasets. The values in parentheses show the corresponding standard deviations.

Model	$\mathrm{WOS_{JTF}}$		$\mathrm{WOS_{JT}}$		$\mathrm{WOS_{CT}}$	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
HPT	84.97 (0.17)	82.13 (0.18)	67.62 (0.18)	61.71 (0.17)	73.25 (**0.12**)	61.87 (**0.15**)
HPTD-ELECTRA	84.75 (0.38)	81.70 (0.42)	67.19 (0.20)	60.91 (0.26)	71.39 (0.39)	58.41 (0.68)
HPTD-DeBERTaV3	85.68 (0.14)	82.93 (0.15)	68.35 (0.12)	62.19 (0.46)	73.45 (0.14)	61.27 (0.56)
$\mathrm{GHLA_{RoBERTa}}$	85.72 (**0.11**)	82.92 (**0.13**)	68.38 (**0.11**)	62.38 (**0.11**)	73.34 (0.25)	61.29 (0.40)

7.5 Conclusion

In this chapter we introduced three new benchmark HTC datasets in the domain of research publications comprising papers' titles and abstracts collected from the Web of Science publication database. Using this data, we created two hierarchical classification datasets that are derived from existing journal-and citation-based classification schemas respectively. These schemas have different disadvantages such as the inaccurate journal-based classifications due to journals that cover research topics that are too broad and the citation-based classifications that only allow a document to belong to a single research field. Therefore, we proposed an approach to combine these two classifications to increase the probability of a document being assigned to the correct classes while allowing documents to be classified into more than one research field. To create the datasets, we sampled documents equally for each of the second-level classes such that our datasets are significantly more balanced than other HTC benchmark datasets. We evaluated the quality of the proposed datasets through a clustering-based analysis and showed that our proposed approach which combines the two classification schemas resulted in documents with the same classes being semantically more similar. Finally, we performed experiments on the three datasets with the best-performing approaches proposed in this thesis along with a baseline approach. We showed that the HPTD-DeBERTaV3 and $\mathrm{GHLA_{RoBERTa}}$ models obtained the best performance across the three datasets.

Chapter 8

Conclusion

In this thesis we investigated the use of transformer-based pre-trained language models (PLMs) for hierarchical text classification (HTC) tasks which have the objective of assigning text documents to a set of classes from a hierarchical class structure. We proposed three new HTC approaches which use various techniques to leverage the language understanding capabilities of PLMs and the hierarchical class structure information in order to improve classification performance.

We performed comprehensive experiments for each of our approaches on three benchmark datasets and compared their performance with recently proposed HTC approaches. Furthermore, we analysed our proposed approaches by evaluating the per-level performance for the levels of the class hierarchy and the performance in low-resource scenarios with only 10% of training data available. Finally, we created three new benchmark datasets in the domain of research publications and evaluated our best-performing approaches on them.

8.1 Summary of Findings and Contributions

In Chapter 4 we proposed the Hierarchy-aware Prompt Tuning for Discriminative language models (HPTD) approach and found that using DeBERTaV3 as the underlying PLM generally improved classification performance over using the ELECTRA PLM. Furthermore, we showed that the HPTD-DeBERTaV3 approach outperformed previous state-of-the-art HTC approaches on two benchmark datasets (WOS and NYT). These results showed that HPTD is an effective HTC approach that is able to efficiently leverage the language understanding capabilities of the PLM through the prompt tuning paradigm with hierarchy-aware prompts. The HPTD approach improves classification performance by reducing the gap between the pre-training and fine-tuning phases through the modification of the input token sequence such that the HTC task resembles the replaced token detection pre-training task used by discriminative language models. Due to the impressive performance of the HPTD approach and other approaches which inject the HTC task into the pre-training task of the PLM, we believe this paradigm is an effective way of using the knowledge obtained during pre-training and that it will become even more valuable as the underlying PLMs improve over time.

Through a comparison of different threshold selection strategies on the HPTD approach, we showed that using a threshold of 0.5 for assigning classes to documents consistently obtained the highest Micro-F1 scores over three benchmark datasets. Furthermore, tuning a separate threshold for each class generally obtained the highest Macro-F1 scores. Therefore, using a threshold of 0.5 is suitable for tasks which favour high accuracy for majority classes while a tuned threshold for each class may provide better results for tasks that require more consistent performance across all of the classes.

In Chapter 5 we evaluated a HTC approach which uses label-wise attention mechanisms and classification layers to fine-tune a PLM on downstream HTC tasks. We investigated several existing attention mechanisms and proposed a new attention mechanism which allows the model to leverage the predictions of all ancestor levels when predicting the classes at a certain level of the hierarchy. We showed that our proposed approach generally outperformed the other attention mechanisms when using BERT as the underlying PLM. Therefore, our approach is able to effectively use the predictions at ancestors levels to guide the classification at a certain level through the proposed hierarchical attention mechanism.

We swapped the BERT model for RoBERTa as the underlying PLM in our approach and showed that it significantly improved performance on two of the benchmark datasets (RCV1-V2 and NYT). Furthermore, using our proposed attention mechanism with the RoBERTa PLM ($\text{GHLA}_{\text{RoBERTa}}$) achieved state-of-the-art performance on the two benchmark datasets with the most complex class hierarchies (RCV1-V2 and NYT). Therefore, we showed that this relatively simple technique for fine-tuning a PLM is an effective approach for HTC tasks and that our proposed attention mechanism is able to leverage the hierarchical class structure to improve classification performance. These results show that label-wise attention mechanisms are well-suited to HTC tasks since they allow the model to learn which features of the text document are the most important for each class separately.

$\text{GHLA}_{\text{RoBERTa}}$ significantly outperformed the HPTD-DeBERTaV3 approach on the RCV1-V2 dataset, achieved comparable results on NYT, and performed slightly worse on the WOS dataset. Even though it is difficult to determine which of these approaches is better in general, we found that $\text{GHLA}_{\text{RoBERTa}}$ performed the most consistently across the three benchmark datasets. This shows that the $\text{GHLA}_{\text{RoBERTa}}$ approach is able to perform well on HTC tasks in different domains with diverse hierarchical class structures.

In Chapter 6 we proposed an approach which uses features extracted from a PLM and a topic model to train a classifier which comprises convolutional layers followed by label-wise attention and classification layers. We found that adding the features obtained by the topic model generally decreased the performance of our classifier. Furthermore, we showed that the proposed approach performs significantly worse than the other approaches proposed in this thesis. The approach proposed in this chapter used the PLM as a feature extractor as opposed to the other approaches proposed in Chapters 4 and 5 which fine-tune the parameters of the PLM for the specific HTC task. Therefore, we believe that the fine-tuning paradigm is an important aspect of effectively leveraging the language understanding capabilities of the PLM for HTC tasks since it allows the pre-trained parameters to be adapted for the specific downstream task. Using the PLM as a feature extractor to train a smaller classifier model is not sufficient for modelling the functional mapping from the input token sequence to the output classifications due to the complexity of these tasks.

Through the low-resource setting results in Chapters 4, 5, and 6 we found that the HPTD-DeBERTaV3 approach obtained the best performance on the WOS and NYT datasets while $\text{GHLA}_{\text{RoBERTa}}$ outperformed the other approaches proposed in this thesis on the RCV1-V2 dataset. Furthermore, the results showed that the approaches proposed in Chapter 5 which use the label-wise attention mechanisms to fine-tune a PLM, have a larger decrease in performance on the WOS and NYT datasets than the HPTD approaches when moving from the full training data to the low-resource setting. Similarly, the HPTD approaches have a larger decrease on the RCV1-V2 dataset. Since the HPTD approaches outperform the label-wise attention approaches from Chapter 5 by large margins on two of the benchmark datasets, we believe that the HPTD approaches are more suitable for low-resource scenarios. The HPTD approach is able to handle low-resource setting effectively since it is designed to leverage the language understanding capabilities of the PLM by learning prompts to extract the existing knowledge of the PLM as opposed to requiring many training instances to make large adjustments to the model parameters.

We found that the Macro-F1 scores consistently have a larger performance decrease than the Micro-F1 scores when moving from the full training data to the low-resource setting. This indicates that the Macro-F1 is a stricter performance metric and that the Micro-F1 may hide information on the performance of the models. This is due to the Micro-F1 averaging the results over all of the instances such that the majority classes bias the results and the results of the minority classes become negligible.

We evaluated the level-wise results of the different approaches proposed in Chapters 4, 5, and 6 along with the average number of training instances for the classes at each level. The results showed that the number of available training instances at a certain level generally has a direct correlation with the performance of the classes at that level, with the classification performance typically decreasing as the number of training instances decrease. From these experiments we showed that the first-level classification performance is high across the three datasets, even for our worst-performing approaches from Chapter 6. We observe that the approaches that perform well overall are those that are able to achieve relatively high classification performance at the lower levels of the class hierarchy, especially in terms of Macro-F1 scores.

Chapter 7 described the methodology for creating three new HTC benchmark datasets with different classification schemas. We proposed an approach to combine journal- and citation-based classifications by filtering out documents and categories that do not have a clear overlap between the two classification schemas. We found that our proposed approach for filtering documents and categories improved the average similarity of documents within a class as well as the separation between documents that are not in the same class. We evaluated our best-performing approaches on the three newly proposed datasets to further analyse their performance and generalisation abilities. We showed that the HPTD-DeBERTaV3 and GHLA$_{\text{RoBERTa}}$ approaches generally obtained the best performance and that their performance was very similar across the three newly proposed datasets.

8.2 Limitations

We evaluated all of our proposed approaches in Chapters 4, 5, and 6 with the three most commonly used benchmark datasets (WOS, RCV1-V2, and NYT). However, the detailed methodology for the creation of two of these datasets (WOS and NYT) is not published. Therefore, there is uncertainty about the accuracy of the true class assignments and it is likely that the datasets contain inaccuracies and omissions in class assignments. Since the results of the proposed approaches depend on the quality of these datasets, the possible inaccurate labels may influence the comparisons between the different approaches. Furthermore, these datasets only cover the domains of news articles and scientific research which are limited in scope and therefore the reported results may not be generalisable to other domains.

For all of the experiments in this thesis, we used the standard Micro-F1 and Macro-F1 evaluation metrics to compare the different classification approaches. However, these metrics may not always be appropriate for comparing hierarchical classification approaches since they ignore the hierarchical class structure during evaluation. Therefore, comparisons between the different approaches may not be comprehensive since we only use these two evaluation metrics.

All of the results were reported using transformer-based PLMs with a `base` size to enable a fair comparison between the different approaches since they use the same number of model parameters. However, the approaches were not compared for PLMs with a lower (`small`) or higher (`large`) number of parameters. The relative performance of the different approaches may vary for differently-sized underlying PLMs, and as PLMs become larger over time, it may become more important to evaluate proposed approaches with larger underlying PLMs to determine whether they improve performance.

8.3 Suggestions for Future Work

The evaluation of hierarchical classification approaches is an area that requires further research since it is still unclear what the best metrics are for evaluating HTC approaches. It seems clear that the Micro-F1 and Macro-F1 metrics hide some information in the evaluation of HTC approaches since they do not consider the hierarchical class structure. For example, should the higher levels of the hierarchy be weighted more during evaluation since it is more important to classify the broader categories correctly? Several alternative evaluation metrics have been proposed [15, 38, 40, 81]. Therefore, to obtain more evidence on the true performance differences of recently proposed HTC approaches, it is important to perform comprehensive experiments with these different evaluation metrics.

A comparison of the state-of-the-art HTC approaches using both smaller and larger underlying PLMs will provide greater insights into the performance and generalisability of these approaches. By comparing these approaches with larger PLMs we may be able to determine which approaches are able to more effectively leverage the intrinsic language understanding capabilities of the PLM. This becomes especially important as the PLMs become bigger and better over time.

The research area of parameter efficient approaches to fine-tuning PLMs for HTC tasks may prove valuable in the future as PLMs become larger over time and, therefore, become increasingly computationally expensive to fine-tune. Therefore, looking at parameter efficient fine-tuning approaches such as LoRA [30] and evaluating their effect on HTC performances is a promising area for future work.

Another topic that deserves a comprehensive study is the use of graph neural networks (GNNs) in HTC approaches. Even though several different approaches have been proposed which use different GNN architectures to encode the class hierarchy [10, 16, 32, 89, 90, 100], a thorough comparison of different architectures and their performance impacts on HTC tasks will be valuable. This would also allow us to establish the relative importance of GNNs in HTC tasks compared to the other components of HTC models.

The explainability of HTC approaches (and deep learning models in general) is an interesting field which will become important when these approaches are applied to sensitive domains such as the healthcare and legal sectors. Several approaches have been proposed to determine which words receive more attention when particular classes are predicted [19, 59, 82], however, further research into how these models leverage the class hierarchy to make decisions may provide valuable insights in terms of explainability as well as the efficacy of incorporating the class hierarchy into the classification process.

List of References

[1] Ba, J.L., Kiros, J.R. and Hinton, G.E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

[2] Bahdanau, D., Cho, K. and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In: *Proceedings of the 3rd International Conference on Learning Representations*. San Diego, California, USA.

[3] Banerjee, S., Akkaya, C., Perez-Sorrosal, F. and Tsioutsiouliklis, K. (2019). Hierarchical transfer learning for multi-label text classification. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 6295–6300. Association for Computational Linguistics, Florence, Italy.

[4] Baumel, T., Nassour-Kassis, J., Cohen, R., Elhadad, M. and Elhadad, N. (2018). Multi-label classification of patient notes: case study on ICD code assignment. In: *Workshops at the thirty-second AAAI conference on artificial intelligence*. New Orleans, Louisiana, USA.

[5] Bengio, Y., Ducharme, R. and Vincent, P. (2000). A neural probabilistic language model. In: *Advances in Neural Information Processing Systems*, vol. 13. MIT Press, Denver, Colorado, USA.

[6] Blei, D. (2012). Probabilistic topic models. *Communications of the ACM*, vol. 55, no. 4, pp. 77–84.

[7] Blei, D., Ng, A. and Jordan, M. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research*, vol. 3, pp. 993–1022.

[8] Blondel, V.D., Guillaume, J.-L., Lambiotte, R. and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. 10008.

[9] Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146.

[10] Chen, H., Ma, Q., Lin, Z. and Yan, J. (2021). Hierarchy-aware label semantics matching network for hierarchical text classification. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pp. 4370–4379. Association for Computational Linguistics, Online.

[11] Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 785–794. San Francisco, California, USA.

[12] Chen, X., Zhang, N., Xie, X., Deng, S., Yao, Y., Tan, C., Huang, F., Si, L. and Chen, H. (2022). KnowPrompt: Knowledge-aware prompt-tuning with synergistic optimization for relation extraction. In: *Proceedings of the ACM Web Conference 2022*, pp. 2778–2788. Association for Computing Machinery, New York, USA.

[13] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1724–1734. Association for Computational Linguistics, Doha, Qatar.

[14] Clark, K., Luong, M., Le, Q.V. and Manning, C.D. (2020). ELECTRA: Pre-training text encoders as discriminators rather than generators. In: *Proceedings of the 8th International Conference on Learning Representations*. OpenReview.net, Addis Ababa, Ethiopia.

[15] Costa, E., Lorena, A., Carvalho, A. and Freitas, A. (2007). A review of performance evaluation measures for hierarchical classifiers. *AAAI Workshop - Technical Report*.

[16] Deng, Z., Peng, H., He, D., Li, J. and Yu, P. (2021). HTCInfoMax: A global model for hierarchical text classification via information maximization. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3259–3265. Association for Computational Linguistics, Online.

[17] Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota, USA.

[18] Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H. and Smith, N. (2020). Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

[19] Dong, H., Suarez-Paniagua, V., Whiteley, W. and Wu, H. (2021). Explainable automated coding of clinical notes using hierarchical label-wise attention networks and label embedding initialisation. *Journal of Biomedical Informatics*, vol. 116, p. 103728.

[20] Gopal, S. and Yang, Y. (2013). Recursive regularization for large-scale classification with hierarchical and graphical dependencies. In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 257–265. Association for Computing Machinery, New York, USA.

[21] Grootendorst, M. (2022). BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv preprint arXiv:2203.05794*.

[22] Hadsell, R., Chopra, S. and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 1735–1742.

[23] Hambardzumyan, K., Khachatrian, H. and May, J. (2021). WARP: Word-level Adversarial ReProgramming. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pp. 4921–4933. Association for Computational Linguistics, Online.

[24] Hayes, P.J. and Weinstein, S.P. (1990). CONSTRUE/TIS: A system for content-based indexing of a database of news stories. In: *Proceedings of the Second Conference on Innovative Applications of Artificial Intelligence*, pp. 49–64. AAAI Press, Washington D.C, USA.

[25] He, K., Zhang, X., Ren, S. and Sun, J. (2016). Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778. Las Vegas, Nevada, USA.

[26] He, P., Gao, J. and Chen, W. (2021). DeBERTaV3: Improving DeBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. *arXiv preprint arXiv:2111.09543*.

[27] He, P., Liu, X., Gao, J. and Chen, W. (2021). DeBERTa: decoding-enhanced BERT with disentangled attention. In: *Proceedings of the 9th International Conference on Learning Representations*. OpenReview.net, Online.

[28] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, vol. 9, pp. 1735–1780.

[29] Hopfield, J.J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558.

[30] Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L. and Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In: *Proceedings of the 10th International Conference on Learning Representations*. Online.

[31] Huang, W., Liu, C., Xiao, B., Zhao, Y., Pan, Z., Zhang, Z., Yang, X. and Liu, G. (2022). Exploring label hierarchy in a generative way for hierarchical text classification. In: *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 1116–1127. International Committee on Computational Linguistics, Gyeongju, Republic of Korea.

[32] Jiang, T., Wang, D., Sun, L., Chen, Z., Zhuang, F. and Yang, Q. (2022). Exploiting global and local hierarchies for hierarchical text classification. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 4030–4039. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates.

[33] Joachims, T. (1998). Text categorization with support vector machines. *Proceedings of the European Conference on Machine Learning*, vol. 10, pp. 137–142.

[34] Johnson, A., Pollard, T., Shen, L., Lehman, L., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. and Mark, R. (2016). MIMIC-III, a freely accessible critical care database. *Scientific Data*, vol. 3, p. 160035.

[35] Kim, Y. (2014). Convolutional neural networks for sentence classification. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1746–1751. Association for Computational Linguistics, Doha, Qatar.

[36] Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In: *Proceedings of the 3rd International Conference on Learning Representations*. San Diego, California, USA.

[37] Kipf, T.N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In: *Proceedings of the 5th International Conference on Learning Representations*. Toulon, France.

[38] Kiritchenko, S., Matwin, S., Nock, R. and Famili, A.F. (2006). Learning and evaluation in the presence of class hierarchies: Application to text categorization. In: *Proceedings of the 19th International Conference on Advances in Artificial Intelligence: Canadian Society for Computational Studies of Intelligence*, pp. 395–406. Québec City, Québec, Canada.

[39] Koller, D. and Sahami, M. (1997). Hierarchically classifying documents using very few words. In: *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 170–178. Morgan Kaufmann Publishers Inc., San Francisco, California, USA.

[40] Kosmopoulos, A., Partalas, I., Gaussier, E. *et al.* (2015). Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Mining and Knowledge Discovery*, vol. 29, pp. 820–865.

[41] Kowsari, K., Brown, D.E., Heidarysafa, M., Meimandi, K.J., Gerber, M.S. and Barnes, L.E. (2017). HDLTex: Hierarchical deep learning for text classification. In: *16th IEEE International Conference on Machine Learning and Applications*, pp. 364–371. Cancun, Mexico.

[42] Kudo, T. and Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71. Association for Computational Linguistics, Brussels, Belgium.

[43] Kurata, G., Xiang, B. and Zhou, B. (2016). Improved neural network-based multi-label classification with better initialization leveraging label co-occurrence. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 521–526. Association for Computational Linguistics, San Diego, California.

[44] LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. *Nature*, vol. 521, pp. 436–44.

[45] LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W. and Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. In: *Advances in Neural Information Processing Systems*, vol. 2, pp. 396–404. Morgan-Kaufmann, Denver, Colorado, USA.

[46] Lewis, D.D., Yang, Y., Rose, T.G. and Li, F. (2004). RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, vol. 5, pp. 361–397.

[47] Li, F. and Yu, H. (2020). ICD coding from clinical text using multi-filter residual convolutional neural network. In: *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 8180–8187.

[48] Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT*, vol. 16, no. 2, pp. 146–160.

[49] Liu, J., Chang, W.-C., Wu, Y. and Yang, Y. (2017). Deep learning for extreme multi-label text classification. In: *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 115–124. Association for Computing Machinery, New York, USA.

[50] Liu, L., Perez-Concha, O., Nguyen, A., Bennett, V. and Jorm, L. (2022). Hierarchical label-wise attention transformer model for explainable ICD coding. *Journal of Biomedical Informatics*, vol. 133, p. 104161.

[51] Liu, P., Qiu, X. and Huang, X. (2016). Recurrent neural network for text classification with multi-task learning. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pp. 2873–2879. AAAI Press, New York, USA.

[52] Liu, W., Pang, J., Li, N., Zhou, X. and Yue, F. (2021). Research on multi-label text classification method based on tALBERT-CNN. *International Journal of Computational Intelligence Systems*, vol. 14, no. 1.

[53] Ma, Y., Liu, X., Zhao, L., Liang, Y., Zhang, P. and Jin, B. (2022). Hybrid embedding-based text representation for hierarchical multi-label text classification. *Expert Systems with Applications*, vol. 187, p. 115905.

[54] Mao, Y., Tian, J., Han, J. and Ren, X. (2019). Hierarchical text classification with reinforced label assignment. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 445–455. Association for Computational Linguistics, Hong Kong, China.

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

[56] McInnes, L., Healy, J. and Astels, S. (2017). HDBSCAN: Hierarchical density based clustering. *Journal of Open Source Software*, vol. 2, no. 11.

[57] McInnes, L., Healy, J., Saul, N. and Großberger, L. (2018). UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, vol. 3, no. 29, p. 861.

[58] Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013). Efficient estimation of word representations in vector space. In: *Proceedings of the 1st International Conference on Learning Representations*. Scottsdale, Arizona, USA.

[59] Mullenbach, J., Wiegreffe, S., Duke, J., Sun, J. and Eisenstein, J. (2018). Explainable prediction of medical codes from clinical text. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1101–1111. Association for Computational Linguistics, New Orleans, Louisiana, USA.

[60] Pal, A., Selvakumar, M. and Sankarasubbu, M. (2020). MAGNET: Multi-label text classification using attention-based graph neural network. In: *Proceedings of the 12th International Conference on Agents and Artificial Intelligence*, vol. 2, pp. 494–505. Valletta, Malta.

[61] Peinelt, N., Nguyen, D. and Liakata, M. (2020). tBERT: Topic models and BERT joining forces for semantic similarity detection. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7047–7055. Association for Computational Linguistics, Online.

[62] Peng, H., Li, J., Wang, S., Wang, L., Gong, Q., Yang, R., Li, B., Yu, P.S. and He, L. (2021). Hierarchical taxonomy-aware and attentional graph capsule RCNNs for large-scale multi-label text classification. *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 6, pp. 2505–2519.

[63] Pennington, J., Socher, R. and Manning, C. (2014). GloVe: Global vectors for word representation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543. Association for Computational Linguistics, Doha, Qatar.

[64] Pudovkin, A.I. and Garfield, E. (2002). Algorithmic procedure for finding semantically related journals. *Journal of the American Society for Information Science and Technology*, vol. 53, no. 13, pp. 1113–1119.

[65] Qin, G. and Eisner, J. (2021). Learning how to ask: Querying LMs with mixtures of soft prompts. In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 5203–5212. Association for Computational Linguistics, Online.

[66] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I. *et al.* (2018). Improving language understanding by generative pre-training.

[67] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W. and Liu, P.J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551.

[68] Reimers, N. and Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 3982–3992. Association for Computational Linguistics, Hong Kong, China.

[69] Robbins, H. and Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400 – 407.

[70] Rose, T., Stevenson, M. and Whitehead, M. (2002). The Reuters corpus volume 1 - from yesterday's news to tomorrow's language resources. In: *Proceedings of the Third International Conference on Language Resources and Evaluation*. European Language Resources Association, Las Palmas, Canary Islands, Spain.

[71] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, vol. 65, no. 6, pp. 386–408.

[72] Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning representations by back-propagating errors. *Nature*, vol. 323, pp. 533–536.

[73] Sandhaus, E. (2008). The New York Times annotated corpus. Tech. Rep., Linguistic Data Consortium, Philadelphia.

[74] Schick, T. and Schütze, H. (2021). Exploiting cloze-questions for few-shot text classification and natural language inference. In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 255–269. Association for Computational Linguistics, Online.

[75] Shi, H., Xie, P., Hu, Z., Zhang, M. and Xing, E.P. (2017). Towards automated ICD coding using deep learning. *arXiv preprint arXiv:1711.04075*.

[76] Shimura, K., Li, J. and Fukumoto, F. (2018). HFT-CNN: Learning hierarchical category structure for multi-label short text categorization. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 811–816. Association for Computational Linguistics, Brussels, Belgium.

[77] Shu, F., Julien, C.-A., Zhang, L., Qiu, J., Zhang, J. and Lariviere, V. (2019). Comparing journal and paper level classifications of science. *Journal of Informetrics*, vol. 13, pp. 202–225.

[78] Silla, C. and Freitas, A. (2011). A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, vol. 22, pp. 31–72.

[79] Stein, R.A., Jaques, P.A. and Valiati, J.F. (2019). An analysis of hierarchical text classification using word embeddings. *Information Sciences*, vol. 471, pp. 216–232.

[80] Strydom, S., Dreyer, A.M. and van der Merwe, B. (2023). Automatic assignment of diagnosis codes to free-form text medical note. *Journal of Universal Computer Science*, vol. 29, no. 4, pp. 349–373.

[81] Sun, A., Lim, E.-P. and Ng, W.-K. (2003). Performance measurement framework for hierarchical text classification. *Journal of the American Society for Information Science and Technology*, vol. 54, no. 11, pp. 1014–1028.

[82] Teng, F., Yang, W., Chen, L., Huang, L. and Xu, Q. (2020). Explainable prediction of medical codes with knowledge graphs. *Frontiers in Bioengineering and Biotechnology*, vol. 8.

[83] Traag, V.A., Waltman, L. and van Eck, N.J. (2019). From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports*, vol. 9, p. 5233.

[84] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u. and Polosukhin, I. (2017). Attention is all you need. In: *Advances in Neural Information Processing Systems*, vol. 30, pp. 6000–6010. Curran Associates, Inc., Long Beach, California, USA.

[85] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. and Bengio, Y. (2018). Graph Attention Networks. In: *Proceedings of the 6th International Conference on Learning Representations*, vol. 6. Vancouver, Canada.

[86] Vu, T., Nguyen, D.Q. and Nguyen, A. (2020). A label attention model for ICD coding from clinical text. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pp. 3335–3341. Yokohama, Japan.

[87] Wang, D., Cui, P. and Zhu, W. (2016). Structural deep network embedding. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1225–1234. Association for Computing Machinery, New York, USA.

[88] Wang, Q. and Waltman, L. (2016). Large-scale analysis of the accuracy of the journal classification systems of Web of Science and Scopus. *Journal of Informetrics*, vol. 10, no. 2, pp. 347–364.

[89] Wang, Z., Wang, P., Huang, L., Sun, X. and Wang, H. (2022). Incorporating hierarchy into text encoder: a contrastive learning approach for hierarchical text classification. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pp. 7109–7119. Association for Computational Linguistics, Dublin, Ireland.

[90] Wang, Z., Wang, P., Liu, T., Lin, B., Cao, Y., Sui, Z. and Wang, H. (2022). HPT: Hierarchy-aware prompt tuning for hierarchical text classification. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 3740–3751. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates.

[91] Werbos, P.J. (1990). Learning representations by back-propagating errors. *Nature*, vol. 323, pp. 533–536.

[92] Wu, J., Xiong, W. and Wang, W.Y. (2019). Learning to learn and predict: A meta-learning approach for multi-label classification. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 4354–4364. Association for Computational Linguistics, Hong Kong, China.

[93] Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K. *et al.* (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

[94] Xiao, L., Huang, X., Chen, B. and Jing, L. (2019). Label-specific document representation for multi-label text classification. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 466–475. Association for Computational Linguistics, Hong Kong, China.

[95] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R.R. and Le, Q.V. (2019). XLNet: Generalized autoregressive pretraining for language understanding. In: *Advances in Neural Information Processing Systems*, vol. 32, pp. 5753–5763. Curran Associates, Inc., Vancouver, Canada.

[96] Yao, Y., Dong, B., Zhang, A., Zhang, Z., Xie, R., Liu, Z., Lin, L., Sun, M. and Wang, J. (2022). Prompt tuning for discriminative pre-trained language models. In: *Findings of the Association for Computational Linguistics*, pp. 3468–3473. Association for Computational Linguistics, Dublin, Ireland.

[97] Yarullin, R. and Serdyukov, P. (2020). Bert for sequence-to-sequence multi-label text classification. In: *Analysis of Images, Social Networks and Texts: 9th International Conference*, pp. 187–198. Springer-Verlag, Berlin, Germany.

[98] Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y. and Liu, T. (2021). Do transformers really perform badly for graph representation? In: *Advances in Neural Information Processing Systems*, vol. 34, pp. 28877–28888. Curran Associates, Inc., Online.

[99] Zhang, T., Wu, F., Katiyar, A., Weinberger, K.Q. and Artzi, Y. (2021). Revisiting few-sample BERT fine-tuning. In: *Proceedings of the 9th International Conference on Learning Representations*. Online.

[100] Zhou, J., Ma, C., Long, D., Xu, G., Ding, N., Zhang, H., Xie, P. and Liu, G. (2020). Hierarchy-aware global model for hierarchical text classification. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 1106–1117. Association for Computational Linguistics, Online.

[101] Zhuang, L., Wayne, L., Ya, S. and Jun, Z. (2021). A robustly optimized BERT pre-training approach with post-training. In: *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, pp. 1218–1227. Chinese Information Processing Society of China, Huhhot, China.