NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATION

BSc THESIS

Hierarchical Large Multi-Label Text Classification of Greek Legal Documents by Utilizing Label Augmentation

Gregory G. Kallinikos

Supervisors: **Manolis Koubarakis,** Professor
Eleni Tsalapati, Supervisor

ATHENS

MARCH 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Κατηγοριοποίηση Ιεραρχικά Δομημένων Ελληνικών Νομικών Εγγράφων πολλών Ετικετών με τη χρήση Επαύξησης Ετικετών

Γρηγόρης Γ. Καλλίνικος

Επιβλέποντες: **Μανόλης Κουμπαράκης,** Καθηγητής
Ελένη Τσαλαπάτη, Επιβλέπουσα

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2022

BSc THESIS

Hierarchical Large Multi-Label Text Classification of Greek Legal Documents by Utilizing Label Augmentation

Gregory G. Kallinikos

S.N.: 1115201500056

Supervisors: **Manolis Koubarakis,** Professor
Eleni Tsalapati, Supervisor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Κατηγοριοποίηση Ιεραρχικά Δομημένων Ελληνικών Νομικών Εγγράφων πολλών Ετικετών με τη χρήση Επαύξησης Ετικετών

Γρηγόρης Γ. Καλλίνικος

Α.Μ.: 1115201500056

Επιβλέποντες: **Μανόλης Κουμπαράκης,** Καθηγητής
Ελένη Τσαλαπάτη, Επιβλέπουσα

ABSTRACT

The aim of this thesis is the development of a hierarchical multi-label text classification model that utilizes the label augmentation technique on a very large dataset with large input sequences. The model will also be handling a shallow hierarchy structure with a very large amount of labels and will be developed with layer-wise guided training [1]. Then, the comparison of that model with other models that do not utilize the same technique will shed light on whether this technique is effective and how much it's worth implementing.

This research bases its roots on the groundbreaking BERT model [2], and consequentially the GreekBERT model [3], as well as the Layer-wise Guided Training for BERT paper of Chalkidis et al., which also introduces the label augmentation mechanism [1]. The use of the transformers library provided by Hugging Face elevates the implementation process and makes the training and evaluation of three different models faster. This library will be the pillar of the development and comparison of our three models from where we will extract our conclusions [4].

The thesis starts with a thorough overview of the path that the NLP research community followed and which lead to the current state-of-the-art architectures, highlighting the advantages and disadvantages of each technique. We believe this is necessary in understanding the key concepts and intricacies of our models and will justify the use of the transformers library and the pre-trained model.

After the historical overview we follow with a detailed presentation of the dataset that will be used. The dataset is called Greek Legal Code (GLC), and it's an openly distributed dataset with a hierarchical structure desirable for our task [5]. The intricacies of the GLC dataset include, among others, the very large input size of the documents, the really high amount of labels in lower hierarchy levels and a high variety of few-shot and zero-shot labels. These prove the difficulty of tackling such a problem and make the results much more concrete and justifiable.

We also make sure to present the details behind the label augmentation technique, as well as the label-wise guided training concept that will be utilized by our primary model. Essentially, our primary model uses certain layers to predict certain hierarchy level labels.

Lastly comes the development and evaluation of our models, which are then compared to each other based on the R-Precision metric. In conclusion, the label augmentation technique, paired with a layer-wise trained model leads to a significant increase of performance without having heavy computational differences compared to our simpler models.

SUBJECT AREA: Natural Language Processing

KEYWORDS: Hierarchical Classification, Label Augmentation, Large Multi-Label Set, Pre-trained Transformers, Greek Legal Code

ΠΕΡΙΛΗΨΗ

Ο σκοπός αυτής της πτυχιακής είναι η υλοποίηση ενός μοντέλου, για ταξινόμηση κειμένων με ιεραρχική δομή και με πολλαπλές ετικέτες. Το μοντέλο αξιοποιεί την τεχνική της επαύξησης ετικετών και η εκπαίδευση γίνεται σε ένα πολύ μεγάλο σύνολο δεδομένων. Η ιεραρχική δομή περιέχει μικρό αριθμό επιπέδων αλλά μεγάλο αριθμό ετικετών και το μοντέλο βασίζεται στην τεχνική της εκπαίδευσης με οδηγό τα επίπεδα του μοντέλου [1]. Στη συνέχεια, η σύγκριση του μοντέλου με άλλα μοντέλα που δέν αξιοποιούν την ίδια τεχνική θα μας αποδείξει την αποτελεσματικότητά της.

Η έρευνα βασίζεται στο μοντέλο BERT [2] και της ελληνικής του μορφής, το GreekBERT [3] καθώς και στο κείμενο πάνω στην οδηγούμενη από τα επίπεδα εκπαίδευσης που επίσης παρουσιάζει την τεχνική της επαύξησης ετικετών [1]. Πολύ βασική είναι η χρήση της βιβλιοθήκης των Transformers μέσω των οποίων φτάνουμε στην εκπαίδευση και σύγκριση των μοντέλων πολύ γρήγορα [4].

Η πτυχιακή ξεκινά με μια αναλυτική παρουσίαση των τεχνικών που χρησιμοποιήθηκαν από τους ερευνητές της Επεξεργασίας Φυσικής Γλώσσας μέχρι τις πλέον πιο διαδεδομένες αναφέροντας τα θετικά και τα αρνητικά τους. Πιστεύουμε πως αυτό είναι απαραίτητο στην κατανόηση της λειτουργίας των μοντέλων που θα υλοποιήσουμε.

Μετά από την ιστορική αναδρομή, συνεχίζουμε με μια αναλυτική παρουσίαση του συνόλου δεδομένων που θα χρησιμοποιήσουμε [5]. Κάποια χαρακτηριστικά του περιλαμβάνουν το μεγάλο μέγεθος των εγγράφων, τον μεγάλο αριθμό των ετικετών και την ύπαρξη πολλών κλάσεων με λίγα έγγραφα ανά κλάση.

Στη συνέχεια, παρουσιάζουμε τις λεπτομέρειες της τεχνικής της επαύξησης ετικετών, καθώς και της εκπαίδευσης με οδηγό τα επίπεδα του μοντέλου, που θα χρησιμοποιηθούν από το βασικό μοντέλο μας.

Εν τέλει, παρουσιάζεται η υλοποίηση των μοντέλων και γίνεται σύγκρισή τους με βάση την μετρική R-Precision. Από τα αποτελέσματα συμπεραίνουμε ότι η τεχνικές που αξιοποιεί το βασικό μοντέλο μας οδηγούν σε αύξηση της απόδοσης, δίχως να έχουμε σοβαρές χρονικές καθυστερήσεις σε σύγκριση με τα πιό απλά μοντέλα μας.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Επεξεργασία Φυσικής Γλώσσας

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Ιεραρχική Ταξινόμηση, Επαύξηση Ετικετών, Νομικά Έγγραφα, Pretrained Transformers, Greek Legal Code

ACKNOWLEDGMENTS

It is of a great pleasure and importance for me to be able to do research in the intriguing and upcoming field of Natural Language Processing as well as that of Artificial Intelligence and combine it with the real-world task in the Law domain, and therefore I am deeply grateful to my professor and supervisor Manolis Koubarakis for providing me with this opportunity. His guidance was absolutely critical in the making of this thesis.

I would also like to thank my research supervisor Eleni Tsalapati, for her patience and constructive critiques during the development of this research work.

CONTENTS

LIST OF FIGURES

LIST OF IMAGES

LIST OF TABLES

PREFACE

The present thesis is a necessary piece in my pursue for the acquisition of a Bachelor's Degree in the Department of Informatics and Telecommunication of the National and Kapodistrian University of Athens.

Implementing and comparing different models is a great way for researchers to find the best practices on the problem at hand and for me personally it has been both a learning experience as well as an intriguing challenge. The fact that it was possible for me to do research in the area of Artificial Intelligence and also pair it with a real life task is what made everything even more exciting.

I was also astounded by the variety of different techniques and depth of knowledge I came across while making this thesis, and I am forever grateful to my professor, Manolis Koubarakis, for providing me with abundant resources as well as a good starting point, and to every researcher in the NLP field that tackles these tasks and tries to find better ways in doing so.

1. INTRODUCTION

For quite a few years Artificial Intelligence has been used to face a numerous amount of problems, among them Natural Language Processing (NLP) problems such as Named Entity Recognition (NER), Part-of-Speech Tagging (POS), Dependency Parsing, Masked Language Modeling and even Text Classification, which is of major interest in this thesis. While a lot of techniques have been introduced, some have established themselves as the state-of-the-art ways to produce not just better results, but also in a faster and more manageable manner [4].

One of these techniques is the use of transformer based models and specifically pretrained BERT models [6] that may be fine-tuned depending on the problem at hand. This has helped many researchers quickly create their own models and reduced the computing cost of training big models [7], and it's essentially what we will be doing in the creation of our models.

The task of Multi-Label Text Classification is essentially the problem of classifying a text sequence to more than just one categories at the same time, and it's a Natural Language Processing task [8]. A bit more challenging problem is the Hierarchical version of Multi-Label Classification [11] where the classes are hierarchically structured and we attempt to predict both the subclass and the corresponding super-classes of the input.

This thesis is about investigating the approach of training and fine-tuning pre-trained transformer based models to predict classes for hierarchically structured legal documents of the Greek Language, while also using an advanced technique called label augmentation [1] on a shallow hierarchy structure. The models that will be presented are themselves based on a pre-trained transformer model called GreekBERT [3], which is itself based on BERT [2] but not fine-tuned for Multi-Label Classification.

The dataset that is used is the Greek Legal Code (GLC) corpus which consists of legal documents written in the Greek Language [5], it presents each legal document labeled in three hierarchy levels; Volume, Chapter, Subject, and it's publicly distributed. Moreover, there will be a comparison of three different models of tackling this problem; firstly with the plain, unmodified model, secondly with the slightly fine-tuned version of the model and lastly with the augmented labels technique which uses a different number of layers of the model to predict the classes. Re-training all of the three models is necessary specifically because the base model was not ever trained for text classification [10]. It is the first time the label augmentation technique is used on such a large dataset, with a low amount of hierarchy levels and a large size of label-ancestors, and researched in a scientific manner.

The results show that by using label augmentation, we can achieve better performance than just the plain model or the slightly fine-tuned one even in the case of the shallow hierarchy and the very large input size of legal documents. Additionally, the comparison of the models show that without any fine-tuning on the specific task, the models' efficiency is way below acceptable and therefore further enhances the importance of making adjustments to the way the model processes its data [12].

The thesis consists of eight chapters, each one of which is listed below with a small introduction:

• *In Chapter 2*, we take a tour on the history of the transformer based models starting from the very beginning, the Word2Vec model and sequence-to-sequence learning, and ending up to the transformers architecture, the GreekBERT and the GreekLegalBERT models.

• *In Chapter 3*, we present the mechanics of pretrained transformer models and BERT as well as the advantages of using such models as a base for implementing new ones.

• *In Chapter 4*, the Greek Legal Code dataset is introduced and discussed in detail.

• *In Chapter 5*, we present the core techniques that will be utilized in the making of our primary model: label augmentation and layer-wise guided training.

• *In Chapter 6*, we see the development and evaluation processes of all of the three models as well as the decisions made in the picking of our optimizer and loss function.

• *In Chapter 7*, we take a look on each models' inner structure along with their evaluation process. In the end we gather the results into a single table for comparison.

Finally, we have *Chapter 8*, where we elaborate on our conclusions and final results.

2. HISTORY OF TRANSFORMER BASED MODELS

Natural Language Processing means enabling computers to make sense of the human language and has been a primary concept of interest in the technological domain for a long time. However, the last decade we have witnessed an unprecedented leap in the way NLP problems handle tasks, mainly enabled by deep learning. The way NLP advanced is of high importance in understanding the current state-of-the-art architecture; attention based models and transformers.

If we look at Machine Learning approaches closely, we see that many of them have focused on Sequence-to-Sequence learning, or Seq2Seq [14]. Seq2Seq essentially is turning one sequence into another sequence. In order for that to happen, recurrent neural networks (RNNs) [15], or more often LSTMs [17] (long short-term memory) or GRUs [18] (gated recurrent unit) has been utilized to avoid the problem of vanishing gradients.

The context for each item is the output from the previous step. The primary components are one encoder and one decoder network. The encoder turns each item into a corresponding hidden vector containing the item and its context. The decoder reverses the process, turning the vector into an output item using the previous output as the input context.

Basically, Seq2Seq learning is to teach a model to map the first sequence into the second, for the example of translation, or the transformation of really long sequences into really short ones, in the case of summarization. Two main branches have been widely used to perform sequence-to-sequence learning; classic Recurrent Neural Networks and Encoder-Decoder models.

In the case of classic Recurrent Neural Networks, the goal is to train a model that can process the sequence item by passing the representation as input, together with the next item from the sequence. In that way, each sequence item is supposedly using context for a word based on the words that have been processed previously. In the case of Encoder-Decoder models, the goal is to train an encoder that can process the input sequence item and turn it into a hidden representation. The decoder then maps that hidden representation into an output sequence.

Encoder-Decoder models have been the base ground for Transformers and they have benefits over Recurrent Neural Networks when used for contextual learning [20]. But why did transformers get to be invented in the first place and how did they get to where they are now?

2.1 Word Embeddings

A very important breakthrough in the field of NLP is the creation of continuous word embeddings, which led to a much more accurate identification of the semantic meaning and the similarity of words. This was presented with the publication of the Word2Vec paper and changed the way word embeddings represented each word [42].

Before Word2Vec, word embeddings used either massive sparse vectors with one-hot encoding, or TF-IDF approaches that ignored common, low-information words and other tokens. These approaches lead to very low accuracy metrics since they do not encode semantic information in their vectors.

With Word2Vec, the training of word embeddings was much more efficient since they contained contextual information; the main concept was that common words should be used together, which makes sense in any language. Based on this concept, the Skip-Gram and Continuous Bag-Of-Words (CBOW) neural network architectures were developed.

Expanding this approach, the sequence-to-sequence model was created, implementing a similar concept into larger sequences of text, like sentences. This was called sequential processing and was the first major milestone in the invention of the transformer models.

2.2 Sequential Processing

Before transformers, most state-of-the-art NLP systems relied on gated RNNs, such as LSTM and Gated Recurrent Units (GRUs). Simple RNNs process tokens sequentially, maintaining a state vector that contains a representation of the data seen after every token [43]. When presented with a new token, the model combines the state representation with the previous tokens. The information of the new token is then added to create a new state that represents the sentence up to the new token [44].

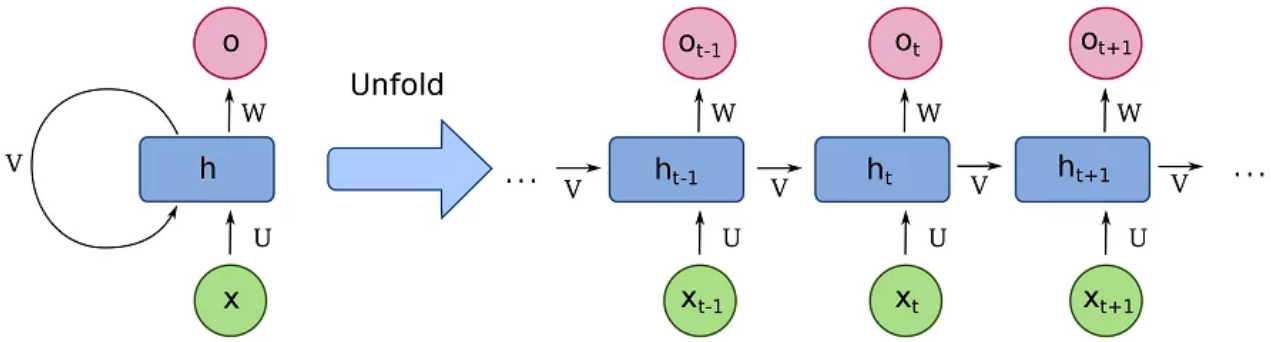

Image 1: A Fully Recurrent Neural Network

Theoretically, the information from one token can propagate arbitrarily far down the sequence, if at every point the state continues to encode contextual information about the token. In practice, the mechanism has its flaw: the vanishing gradient problem, which essentially leaves the model's state at the end of a long sentence without much precise, extractable information about preceding tokens.

The dependency of token computations on results of previous token computations also makes it hard to parallelize computation on modern deep learning hardware. This can make the training of RNNs inefficient. The vanishing gradients in the back-propagation step of computing the error backwards when input sequences are long causes this problem. While they show good results for shorter sequences, in longer ones the distance between the relevant words can be too long for results to be acceptable. This is where LSTM came into place.

2.3 LSTM (Long Short-Term Memory) Models

LSTM stands for Long Short-Term Memory and it's a deep learning system that avoids the vanishing gradients problem [21]. LSTM models are normally augmented by recurrent gates called "forget gates". LSTMs prevents back-propagated errors from vanishing or exploding gradients. Instead, the errors can flow backwards through unlimited numbers of virtual layers unfolded in space. For that reason, LSTMs are better in learning tasks when the distance between the relevant words is big [30].

In practice, by utilizing LSTM units a model can partially solve the vanishing gradient problem, because LSTM units allow gradients to also flow unchanged. Unlike standard feed-forward neural networks, LSTM has feedback connections. A common LSTM unit is composed of a cell, an input gate, an output gate, and a forget gate.

LSTM networks are good candidates for classifying, processing and making predictions based on time series data [30]. They were developed to deal with the vanishing gradient problem that traditional RNNs encounter.

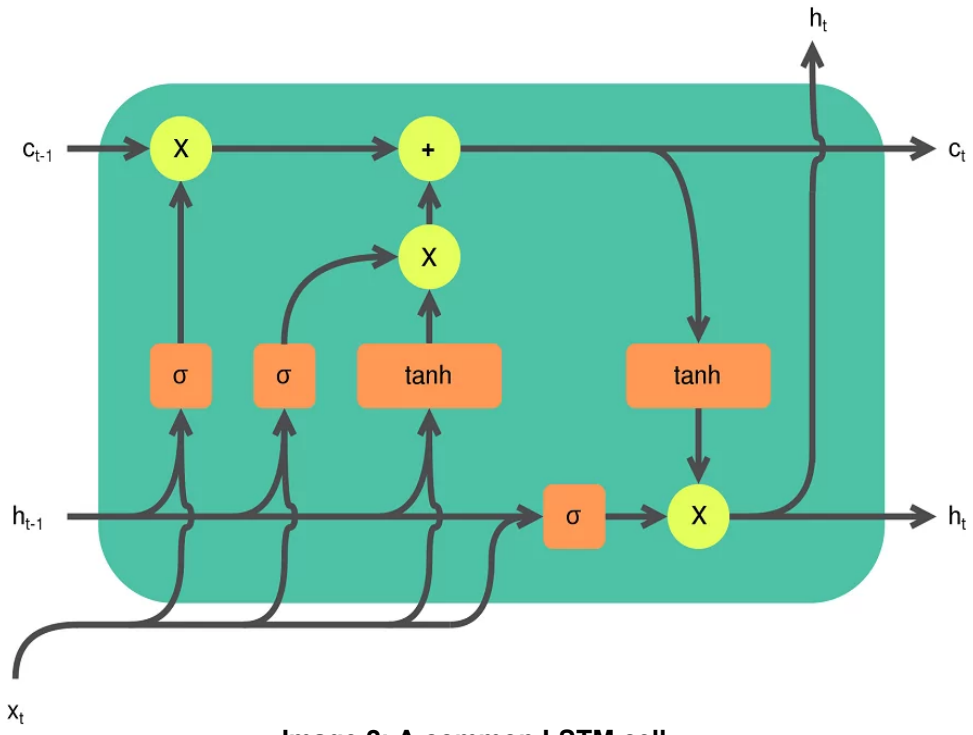

Image 2: A common LSTM cell

The h_{t-1} cell is essentially the forget gate and as presented in Image 2, its output is passed through a Sigmoid function, together with the input X at t, X_t . By doing so, the network can "forget" certain aspects from the cell state based on the current input values and the previous hidden state through this gate. In a sense, LSTM utilizes cell outcomes of what must be kept and what must be forgotten to produce an output.

Many applications use stacks of LSTM RNNs to find an RNN weight matrix that maximized the probability of the label sequences in a training set, given the corresponding input sentences. Ultimately, LSTM models can learn to recognize context-sensitive languages and similar concepts.

During training, LSTM units diminish the vanishing gradients problem by keeping the error in the LSTM units' cell as the error values are back-propagated from each output layer [17]. This way, the error is continuously provided back to each of the LSTM units' gates until they learn to cut off the value.

However successful they might have been, LSTM networks can still suffer from the exploding gradient problem, but they are still used and applied in many real life problems.

2.4 GRU (Gated Recurrent Unit) Models

GRU stands for "Gated Recurrent Unit" and essentially is a simplification of Long Short-Term Memory networks. The key components of a GRU is both an input gate and a forget gate, essentially missing the output gate of the LSTM system. Consequentially, they are much faster in terms of training speed than an LSTM, due to having fewer parameters [18]. They have been shown to exhibit better performance on certain smaller and less frequent datasets.

There are several variations on the full gated unit, with gating done using the previous hidden state and the bias in various combinations, and a simplified form called minimal gated unit.

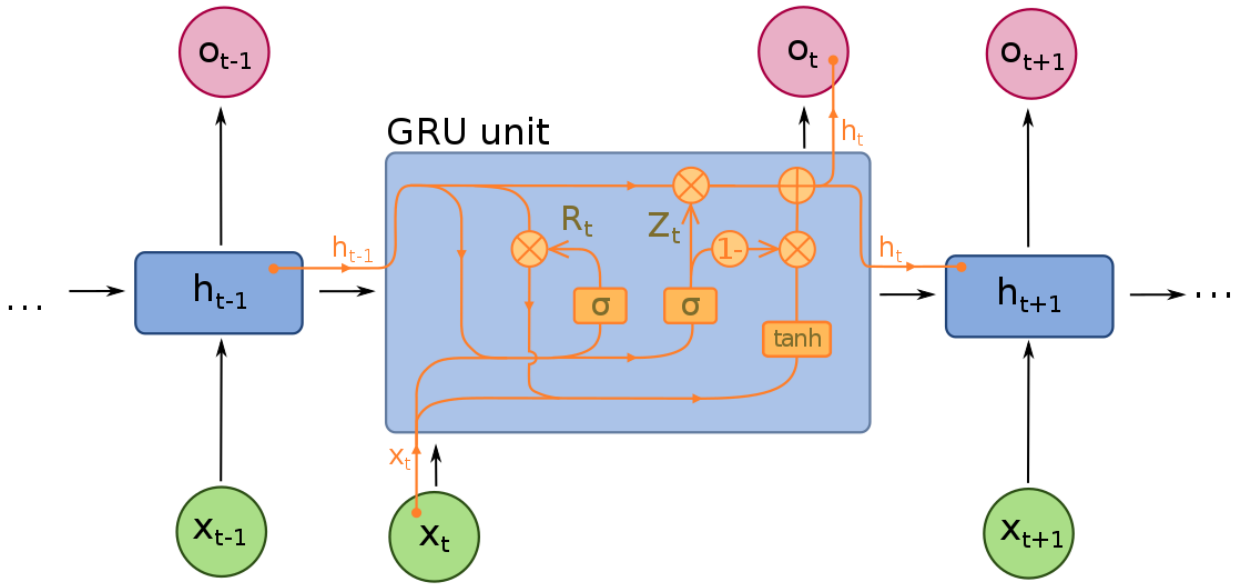

Image 3: A common Gated Recurrent Unit

By means of benefits in terms of backwards error computation, LSTMs and GRUs have yielded better performance over classic, vanilla RNNs. However all these models face the same problem produced when processing long sequences of data. This problem arises because the memory is updated by means of a short-term change: in LSTMs, the memory is adapted based on short-term interrelationships. This means that longer-term ones, while they do pass through memory, they are forgotten over time.

While both LSTM and GRU models were considered a big breakthrough at their time, they still faced the problems of exploding gradients and of having to read a sequence either left-to-right or right-to-left, making them difficult to use in NLP tasks where context can be inferred in both ways. For this reason, the attention mechanism was invented [33].

2.5 Attention Mechanism

In neural networks, attention is a technique that mimics cognitive attention. The effect enhances some parts of the input data while diminishing other parts. That means that the network can shift its focus to what we think is important and it is particularly effective in long sequences where a small part is of bigger significance. Learning which part of the data is more important than others depends on the context and is trained by gradient descent. Essentially, the attention mechanisms allow a model to draw from the state of any preceding point along the sequence, however long [20]. The attention layer can access all the previous states and weight them according to a taught measure of relevancy, providing relevant information about far-away tokens.

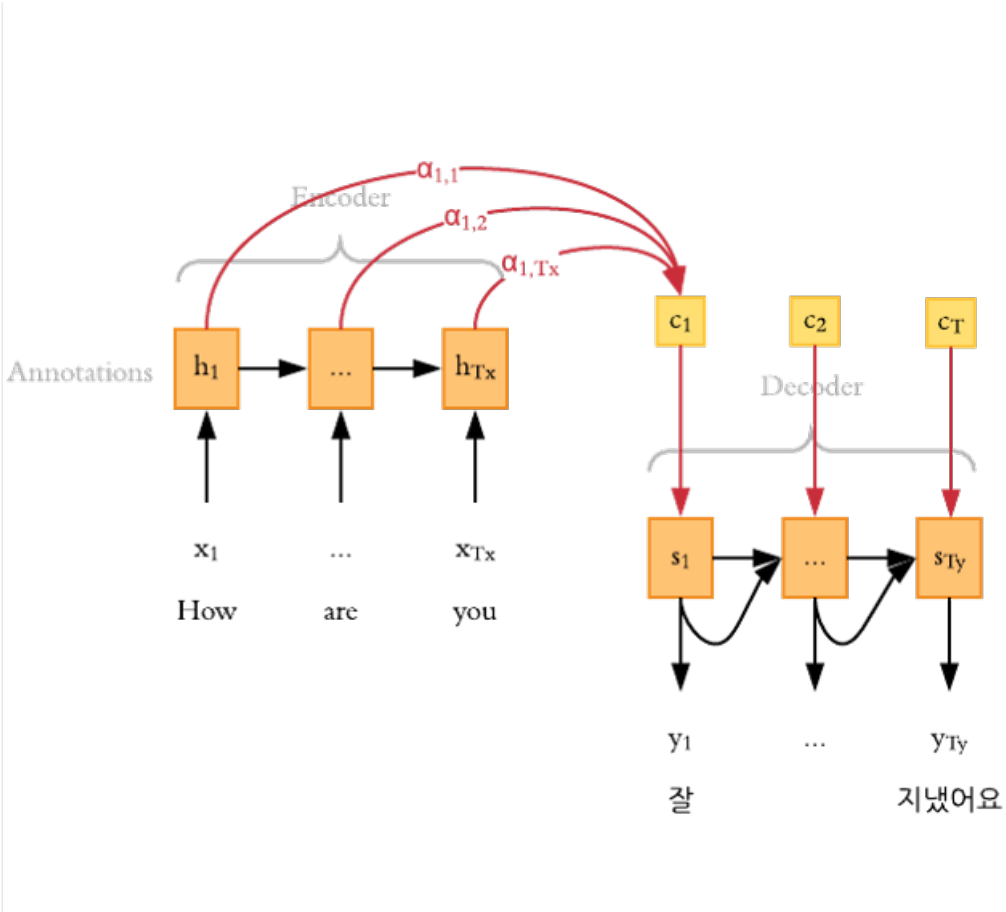

Image 4: A brief illustration of the attention mechanism, translating English to Korean

For example, in the case of language translation, context is essential to assign the meaning of a word in a sentence. Many times a first word of one language can be the last word in the translation to another. In a classic LSTM model, in order to produce that last word, the model is given only the state vector of the first word in the starting language. An attention mechanism can be added to address this problem: the decoder is given access to the state vectors of every input word, not just the last, and can learn attention weights that dictate how much to attend to each starting input state vector [33].

The encoder-decoder architecture is interesting to take a look at, but essentially this still was not enough and LSTMs and GRUs all faced the same same bottleneck: even though the memory itself was improved through attention, computation was not. The

process is still sequential, because each token has to be processed in left-to-right or right-to-left order.

In conclusion, the attention mechanism has been utilized to increase performance, specifically accuracy and bleu score, which is basically evaluation score in language translation [34]. However, it is time consuming and can be hard to parallelize. The attention weights address the explainability problem that common neural networks had, adjusting their focus according to text context.

2.6 Transformer Models

A transformer is a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data. Transformers involve an encoder segment and a decoder segment and they are increasingly the model of choice for NLP problems, replacing LSTMs and GRUs. The core attribute of transformers is that they do not necessarily process the data in order. Rather, the attention mechanism provides context for any position in the input sequence.

For example, if the input data is a natural language sentence, the transformer does not need to process the beginning of the sentence before the end. Rather, it identifies the context that confers meaning to each word in the sentence.

This feature allows for more parallelization than RNNs and therefore reduces training times. This allows training on larger datasets than what was possible with just LSTMs and the transformer models became state-of-the-art when the groundbreaking development of pretrained systems such as BERT (Bidirectional Encoder Representations from Transformers) was introduced [2]. These models, having been trained on large language datasets, such as the Wikipedia Corpus and Common Crawl, and can be fine-tuned for specific tasks.

The original Transformer model uses an encoder-decoder architecture. The encoder part, which can be identically repeated, each time increasing the precision of the encoding. It consists of a multi-head attention segment, essentially being the self-attention mechanism, and a feed forward neural network. The self-attention mechanism accepts input encodings from the previous encoder and weights their relevance to each other to generate output encodings [40]. The multi-head attention blocks are subsequently being added together, and the layer is then normalized with the residual input. The feed-forward neural network further processes each output encoding individually. These output encodings are then passed to the next encoder as its input, as well as to the decoders eventually.

The first encoder takes positional information and embeddings of the input sequence as its input, rather than encodings.

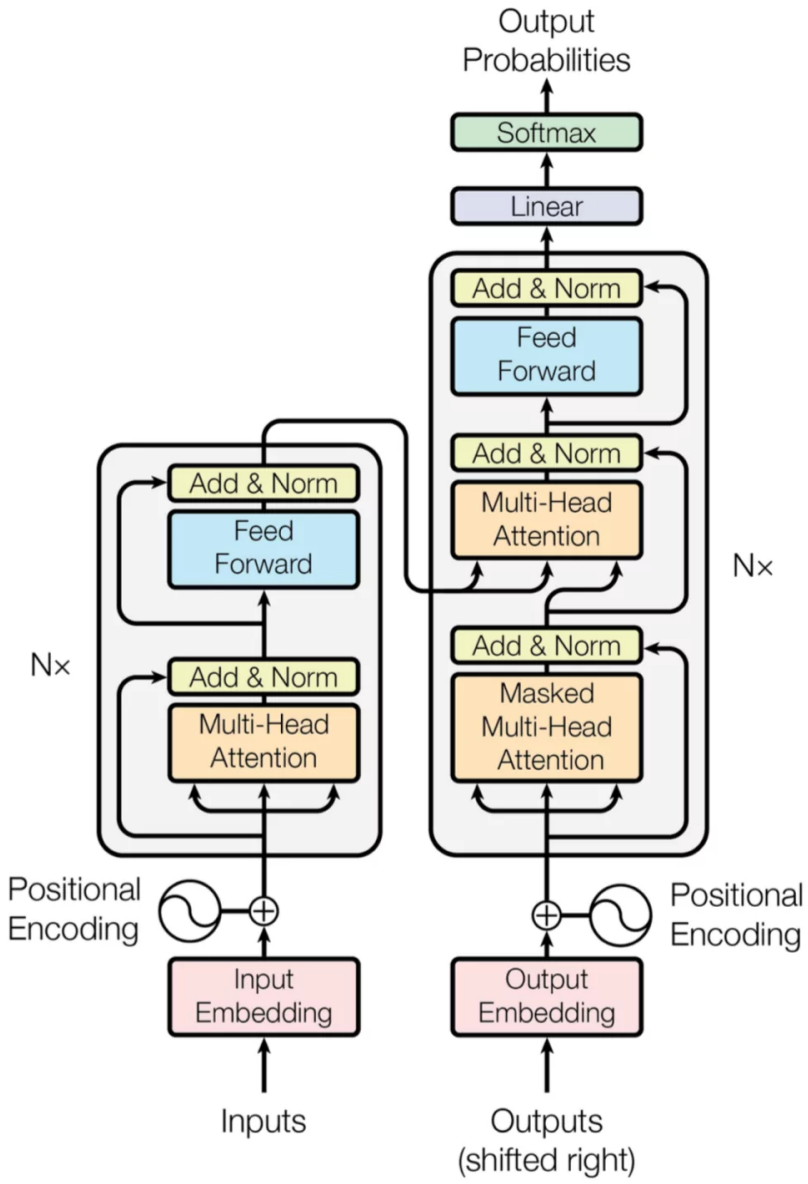

Image 5: A typical Transformer architecture

As presented in Image 5, each decoder consists of three major components: a masked multi-head attention segment, which represents the self-attention mechanism, another attention mechanism over the encodings, and lastly a feed-forward neural network. Like the first encoder, the first decoder takes positional information and embeddings of the output sequence as its input, rather than just the encodings. The output sequence is partially masked to prevent a reverse information flow that is produced when the transformer uses the current and future output to predict an output. The last encoder is followed by a last linear transformation and Softmax layer, to produce the output probabilities over the vocabulary.

This way of working has allowed Natural Language Processing practitioners to achieve extremely impressive results in terms of text processing while also solving the issues with long-term memory and computation speed. Transformers typically undergo semi-supervised learning involving unsupervised pretraining followed by supervised fine-tuning. Pretraining is typically done on a larger dataset than fine-tuning, duo to the limited availability of labeled training data. While some systems still use LSTMs and GRUs, Transformer based models have become state-of-the-art, starting from the groundbreaking BERT development.

2.7 GreekBERT

The GreekBERT model is essentially a monolingual language model for Greek, that has its core architecture based on the BERT model. What this means is that it utilizes the pretrained BERT-BASE model structure and it is trained on Greek corpora so that a monolingual version of BERT is available for researchers to experiment and apply it on Greek data. The GreekBERT model is the base of all of our models in this thesis and is described in exhaustive detail in the paper *'GreekBERT: The Greeks visiting Sesame Street' [3]*. There are two stages in employing the GreekBERT model:

1. Pretraining BERT with the Masked-Language Modeling (MLM) and Next-Sentence Prediction (NSP) objectives on Greek corpora.
2. Fine-tuning the pretrained BERT model for downstream Greek NLP tasks.

The development of GreekBERT follows the notion that monolingual BERT-based language models very often outperform their multilingual counterparts, and as presented in the paper, the GreekBERT model always yields better results when used with Greek language as input.

The evaluation of the GreekBERT model was conducted on three NLP tasks; part-of-speech tagging (POS), named entity recognition (NER) and natural language inference (NLI). The performance recorded showed the significant improvement over two multilingual Transformer-based models, namely M-BERT and XLM-R, as well as shallower neural baselines of pretrained word embeddings, which accounted up to 5%-10%.

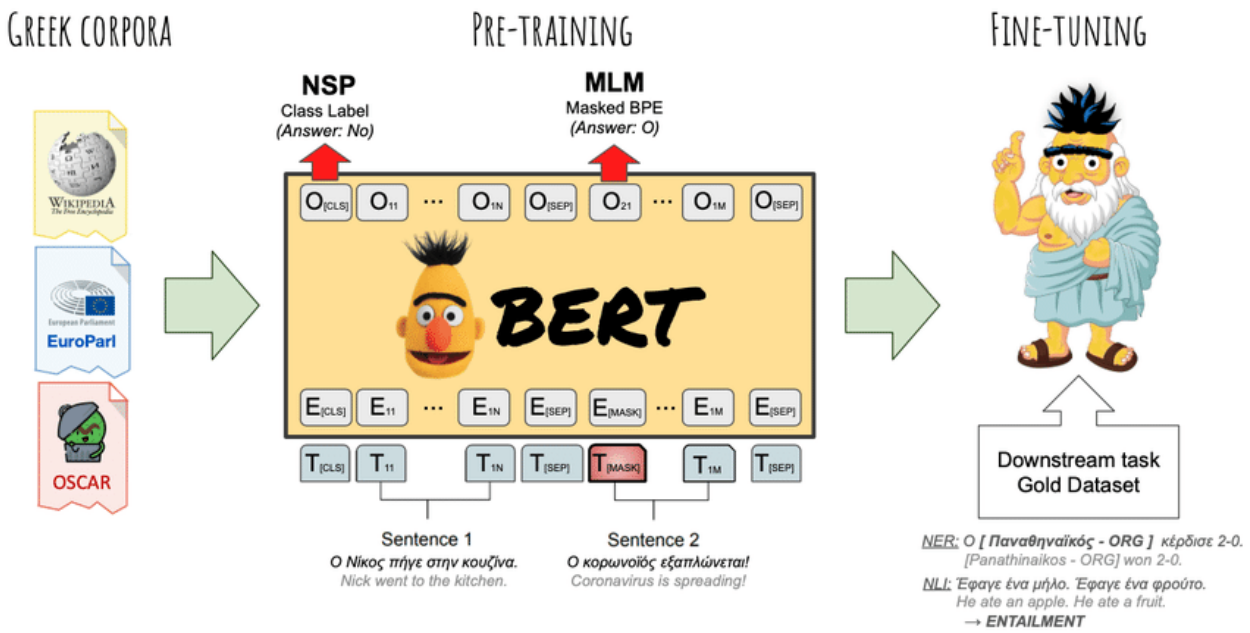

Image 6: The two stages of employing GreekBERT

This massive improvement establishes the GreekBERT model as the state-of-the-art model when dealing with text written in the Greek language.

At the moment the paper was being written, the publicly available resources for Greek NLP were very limited and even more importantly, there has not been any Transformer-based pretrained language model research for Greek. The GreekBERT model was trained on 29 GB of text from Greek corpora from (a) the Greek part of Wikipedia, (b) the Greek part of the European Parliament Proceedings Parallel (Europarl) and (c) the Greek part of OSCAR, a clean version of Common Crawl.

Later on, the detailed comparison of the GreekBERT model with other models prove that the GreekBERT model achieves the aforementioned 5%-10% increase in performance compared with other models. Overall, the development of the GreekBERT model really boosts the NLP research and applications for Greek and their work is essentially a core pillar in the making of this thesis.

2.8 GreekLegalBERT

The GreekLegalBERT model was another significant breakthrough in the Greek NLP research domain. Its development is thoroughly presented in the thesis of Konstantinos I. Athinaios, 'Named Entity Recognition using a Novel Linguistic Model for Greek Legal Corpora based on BERT model' [38]. The GreekLegalBERT model essentially utilizes the pretrained GreekBERT model and further enhances it so that it can perform better when presented specifically with Greek legal documents.

There have been three main pillars in the development of the GreekLegalBERT model:

1. The innovative BERT model of Google, which has been the core in the development of many pretrained models.
2. The Hellenized version of BERT, namely GreekBERT which was presented in the previous part.
3. The master's dissertation of Iosif Angelidis, which also concerns the recognition of named entities on legal data.

The GreekLegalBERT model is trained on Legal corpora and then compared with other models, as well as the GreekBERT model. The exact corpora used is the Greek Legislation available in the Nomothesi@ platform which was then pre-processed to be appropriate input for the BERT model.

After the necessary pre-processing, the GreekLegalBERT model was developed by continuous training and evaluation on the aforementioned corpora. Lastly, the model was further tested on the Masked Language Modeling task and then compared with the GreekBERT model on the Named Entity Recognition task.

In the results presented, both models perform similarly, with marginal variations in performance, where sometimes the GreekLegalBERT model prevails, and some other times the GreekBERT model prevails. Nevertheless, this shows that fine-tuning on Greek corpora increases overall performance and is a significant finding for the Greek NLP research community.

The important thing to note here, is that the creation of GreekLegalBERT was based on the pretrained BERT model, which shows just how much can be achieved by the correct planning and utilization of pretrained transformer models and BERT.

2.9 Summary

In this chapter we presented the main path the NLP researchers took in the development of the current state-of-the-art NLP models. We started from the very beginning, the Word2Vec embeddings, and ended all the way up to the transformers, and the creation of the GreekBERT and GreekLegalBERT models. In the next chapter, we will describe in detail the procedures and inner mechanisms of the pre-trained transformer models and BERT.

3. PRE-TRAINED TRANSFORMERS & BERT

Pre-trained transformer models, like BERT (Bidirectional Encoder Representations from Transformers) [2] and GreekBERT [3] which will be used later on, are pre-trained state-of-the-art models that are provided to be used on numerous tasks, and especially NLP tasks. The models are trained on large datasets and then their weights are saved to be used by any other interested researcher, which introduces an easy-to-use entry access point. Additionally, they are easy to train and evaluate and the computational costs are generally low.

These models use a very similar practice that has been utilized in image recognition, called Transfer Learning where essentially a model is trained on a massive dataset of labeled images and then this model could be downloaded to be used and trained in other, usually smaller datasets. This way, researchers would not have to start from scratch to get their model to deal with image recognition tasks like face recognition.

By using an existing solution and restructuring it can help into getting better results at a faster rate and reduce training time significantly. For this thesis, we will be using Hugging Face transformers which provide a wide variety of NLP libraries and tools that make the development much simpler. The steps when using a pre-trained transformer are quite simple in theory when compared with having to build the complex architecture of the Transformer models from scratch.

A typical pipeline for a pre-trained transformer model is shown in the following figure.

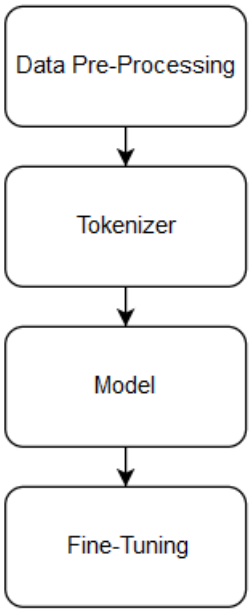

Figure 1: Steps for a pre-trained model

The development starts with the data pre-processing part where the data is usually modified so that only the important parts are kept. Then, the input is passed into a

predefined Tokenizer, and notably our model will be using the same tokenizer of GreekBERT, extracted directly from the Hugging Face library. Then the actual model is loaded and it may be used as is or fine-tuned as we may want. The fine-tuned model can be further trained into new datasets keeping the initial parameter weights.

BERT was introduced as a deep language representation model based on Transformers and is designed by pre-training deep bidirectional representations from unlabeled text using self attention. Since the release of BERT, many BERT based models have become state-of-the-art, essentially because of the powerful concept of attention which can be parallelized and therefore the model can instantly read the input sequence and infer the meaning of a token, based both on left and right context.

At its core BERT is a transformer language model with a variable number of encoder layers and self-attention heads. The architecture is “almost identical” to the original transformer implementation. BERT was pretrained on two tasks: language modeling, using masked tokens and predicting them from context, and next sentence prediction, if a chosen next sentence was probable or not given the first sentence. BERT can be fine-tuned with less resources on smaller datasets to optimize its performance on specific tasks.

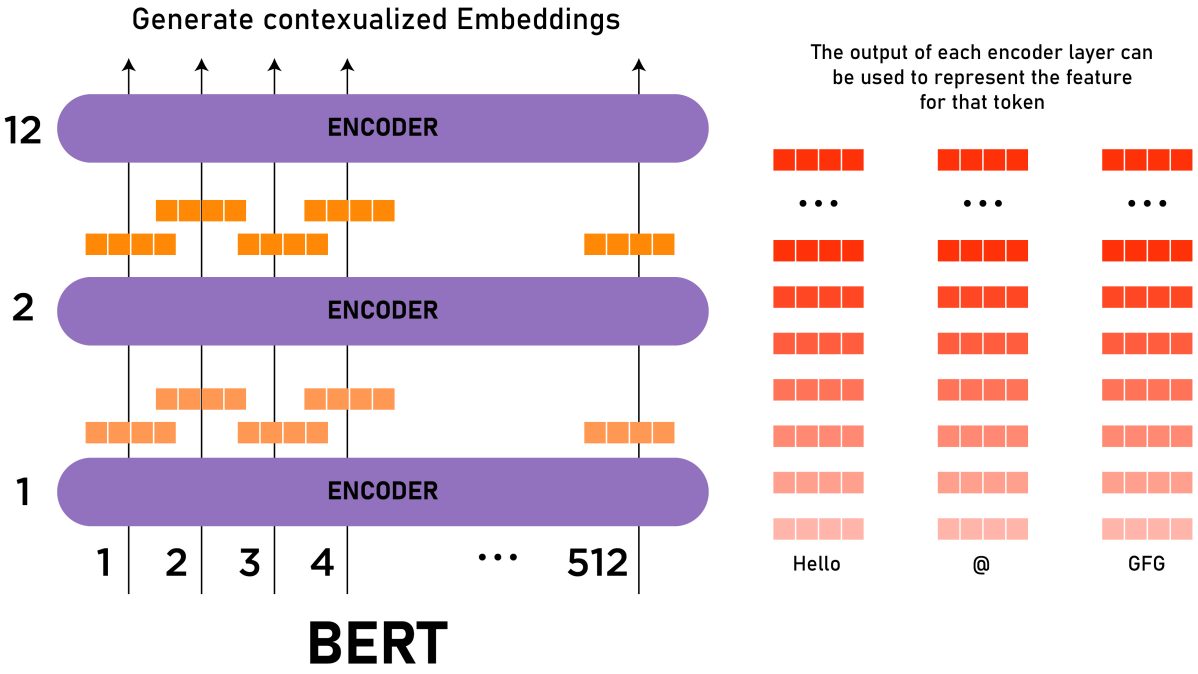

Image 7: BERT encoders and embeddings architecture

Generally, BERT is very similar to the transformer model, but it has some few key differences. Basically, BERT uses the typical encoder part of a Transformer model to encode semantic and syntactic information in the embedding, which is needed for many tasks, but on the other hand it does not use the decoder part, so the output part is actually an embedding and not text. This enables a lot of functionality for each specific task, for example embeddings can be compared with cosine similarity. Finally, BERT uses two training techniques, namely Masking and Next Sentence Prediction.

When BERT was published, it achieved state-of-the-art performance on a number of natural language understanding tasks and has been used as base for many other

models. One of these is the GreekBERT model which will be the base for the models constructed in this thesis. The advantage of such models is two-fold: firstly they have already been trained on a very large dataset, which is by itself a very slow and tedious process that can take many days. Secondly, these models can be fine-tuned and further enhanced depending on the specific task at hand. For BERT specifically we need just a few thousand examples on our data for our fine-tuning to produce really good results.

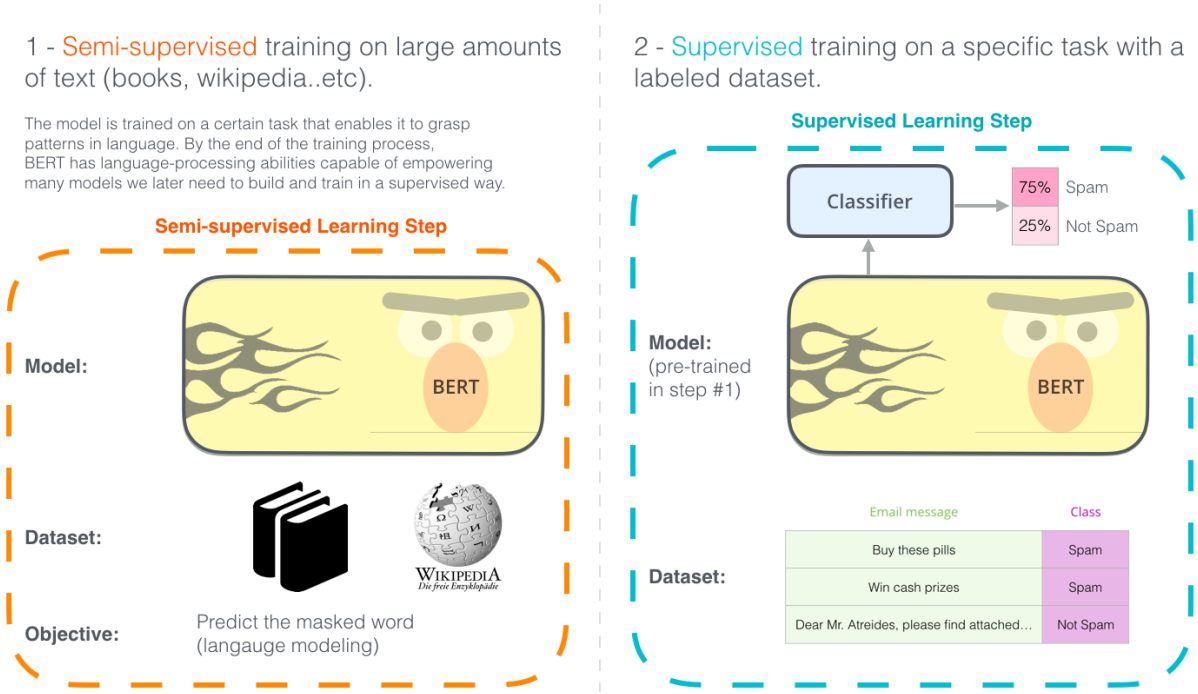

Image 8: The two steps of how BERT is developed. You can download the model pre-trained in step 1 (trained on un-annotated data), and only worry about fine-tuning it for step 2

The GreekBERT model essentially is a pre-trained and fine-tuned version of BERT designed specifically for the Greek language and, as expected from monolingual models, achieves state-of-the-art performance in the NLP tasks of Named Entity Recognition, Part-of-Speech Tagging and Natural Language Inference when used on Greek datasets. Since this model has not been directly fine-tuned for Text Classification, the re-training of the model is absolutely needed in order to produce considerate performance results. Since the dataset being used is a collection if Greek Legal documents, GreekBERT is a better starting point than BERT.

BERT-like models essentially use a similar concept as transfer learning, where they allow us to build on already acquired knowledge and more importantly we can add our own layers on top of these models while freezing the parameters of the last layer or we can fine-tune our BERT-like model by letting specific higher layers as unfrozen. Even though we do not need a very large dataset size, we will be using the whole training set of the Greek Legal Code corpus, which is thoroughly presented in our following chapter.

4. GREEK LEGAL CODE DATASET

The Greek Legal Code (GLC) corpus is an openly distributed dataset on Hugging Face consisting of legal resources from Greek legislation and it's classified into three multi-level categories, starting from broader to narrower fields [37]. This catalog is a thorough representation of the Greek Legislation, with resources since the creation of the Greek state.

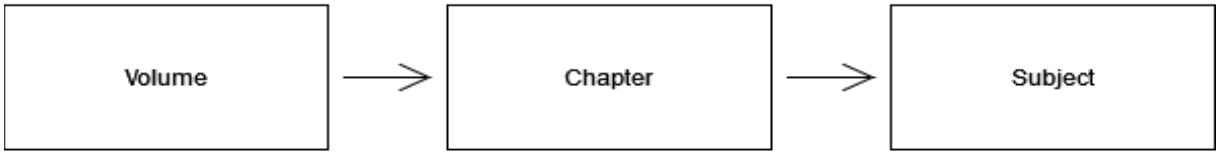

Figure 2: GLC Label Hierarchy Levels

The GLC consists of 47 legislative volumes and each volume corresponds to a main thematic topic. Each volume is divided into thematic sub categories which are called chapters and are in total 389, and subsequently, each chapter breaks down to subjects which contain the legal resources, counting up to 2285 subjects. This introduces a shallow hierarchy structure consisting of three levels.

The dataset is conveniently split into training, development and test sets where all documents are distributed equally for all levels of the class hierarchy so that a much more fair split is introduced. As for the label frequency, only some classes are under-represented, but increasing in number as we go down in the hierarchy levels. For the volume level for example, all classes belong are frequently represented and there are not any few-shot or zero-shot classes, while for the subject level the data contains more few-shot classes.

Table 1: Distribution of labels in each hierarchy level

Level	Total	Frequent	Few-Shot (<10)	Zero-Shot
Volume	47	47	0	0
Chapter	389	333	53	3
Subject	2285	712	1431	142

The shallow hierarchical structure of the dataset creates the opportunity of testing the label augmentation technique when presented with just a few hierarchical levels. Moreover, we do not have to truncate our hierarchy levels which is convenient, but we have to make sure we handle the few-shot and zero-shot labels correctly. This means

that we will be doing predictions for all of the labels in each level and we can expect really low performance scores in our early models.

The documents are provided in a very simple and useful manner of two fields:

1. The *text* field which contains the full content of each document. This content needs to be pre-processed correctly if we want to make accurate predictions since it comes with the full header of the legal document, meaning it has a lot of categorical arithmetic values that do not provide anything to our predictions.
2. The actual *label* that denotes the class in which the document belongs for the corresponding volume, chapter or subject level.

Table 2: Data splits for each set

Split	No of Documents	Avg. words
Train	28536	600
Development	9511	574
Test	9516	595

We can also take a look into the number of documents per each set and appreciate how much work has been done for the creation of the dataset. Later on, we will see that we have to truncate the documents to 512 tokens, since that is the maximum size of the input that can be provided to a BERT-like model, and consequentially the GreekBERT model that will be used as a base for all of our models.

Loading and using the Greek Legal Code dataset is very simple and we have to make sure to load each hierarchy level separately and eventually concatenate the labels in order for each document to be assigned with all three labels from all the three hierarchy levels.

```
volume_dataset = load_dataset("greek_legal_code", "volume")
chapter_dataset = load_dataset("greek_legal_code", "chapter")
subject_dataset = load_dataset("greek_legal_code", "subject")
✓ 26.8s                                                    Python
```

Image 9: Loading the datasets for each level

```
train_text = volume_dataset['train']['text']
train_labels = volume_dataset['train']['label']
validation_text = volume_dataset['validation']['text']
validation_labels = volume_dataset['validation']['label']
test_text = volume_dataset['test']['text']
test_labels = volume_dataset['test']['label']
✓ 3.1s
```

Image 10: Extracting the text and label fields from the volume dataset, for each of its subsets

```
    pprint(train_text[1])
    pprint(train_labels[1])
✓ 0.2s
('15. ΝΟΜΟΘΕΤ.ΔΙΑΤΑΓΜΑ υπ’ αριθ.262 της 8/8 Ιουλ.1941 Περί συστάσεως θέσεως '
 'Γεν. Επιθεωρητού των εν Πελοποννήσω Νομαρχιών και αναστολής της ισχύος των '
 'περί εντοπιότητος διατάξεων.  Προσωρινής ισχύος, ως και τα: α)Ν.Δ.794 της 29 '
 'Νοεμ./12 Δεκ.1941 περί συστάσεως εν Βόλω Γεν. Επιθεωρήσεως Νομαρχιών και '
 'συμπληρώσεως του άρθρ.86 Α.Ν. 1488/1938.  β)Ν.Δ.1214 της 23 Μαρτ./16 '
 'Απρ.1942 περί συμπληρώσεως του υπ’ αριθ.794/1941 Ν.Δ/τος, άτινα '
 'εκωδικοποιήθησαν δια του  γ)Κ.Δ.21 Ιουλ./1 Οκτ.1942 περί συστηματικής '
 'κατατάξεως, κωδικοποιήσεως κλπ. των περί Γεν. Επιθεωρητών Νομαρχιών κειμένων '
 'διατάξεων. Σχετικός και ο: δ)Νόμ.207 της 29 Μαίου/1 Ιουν.1943 περί '
 'λειτουργίας, συντηρήσεως κλπ. των εξ επιτάξεως αυτοκινήτων και περί τρόπου '
 'αυξήσεως των παγίων προκαταβολών δι’ έξοδα μετακινήσεως  των Γεν. '
 'Επιθεωρητών Νομαρχιών.  Αι θέσεις των Γεν. Επιθεωρητών κατηργήθησαν δια του '
 'Νόμ.35/1944 (κατωτ. αριθ. 26).  ')
35
```

Image 11: Example of a legal document, text and label

By inspecting the datasets, we can see their structure in code. On the upper level, each dataset (volume, chapter, subject) is split into train, validation and test sets. Then, each of these aforementioned sets are further split into the text and label fields, making it simple to extract this information and process it in our own environment. So, as presented above, we can see the actual *text* part of our legal documents, as well as access their corresponding label. In this example, the document with the presented text is given the label 35.

We can now also see the abundant amount of arithmetic values and punctuation that need to be truncated since they do not offer any valuable information about the meaning of the document. Some documents have much longer text field, but with the truncation of meaningless tokens we will keep all the necessary information and then our tokenizer will keep 512 actually meaningful tokens. This will lead to a much better and correct training and evaluation process and eventually yield much better results compared to a technique that would keep these random numeric tokens.

We can get the actual name of the labels from the HuggingFace GLC dataset repository, as given in the *greek_legal_code.py* file. For example, some label names for the volume dataset is presented below, noting that the total amount of labels would take a lot of pages to present. However, we can now link each label number with the actual label title.

```
_LABEL_NAMES = {
    "volume": [
        "ΚΟΙΝΩΝΙΚΗ ΠΡΟΝΟΙΑ",
        "ΓΕΩΡΓΙΚΗ ΝΟΜΟΘΕΣΙΑ",
        "ΡΑΔΙΟΦΩΝΙΑ ΚΑΙ ΤΥΠΟΣ",
        "ΒΙΟΜΗΧΑΝΙΚΗ ΝΟΜΟΘΕΣΙΑ",
        "ΥΓΕΙΟΝΟΜΙΚΗ ΝΟΜΟΘΕΣΙΑ",
        "ΠΟΛΕΜΙΚΟ ΝΑΥΤΙΚΟ",
        "ΤΑΧΥΔΡΟΜΕΙΑ - ΤΗΛΕΠΙΚΟΙΝΩΝΙΕΣ",
        "ΔΑΣΗ ΚΑΙ ΚΤΗΝΟΤΡΟΦΙΑ",
        "ΕΛΕΓΚΤΙΚΟ ΣΥΝΕΔΡΙΟ ΚΑΙ ΣΥΝΤΑΞΕΙΣ",
        "ΠΟΛΕΜΙΚΗ ΑΕΡΟΠΟΡΙΑ",
        "ΝΟΜΙΚΑ ΠΡΟΣΩΠΑ ΔΗΜΟΣΙΟΥ ΔΙΚΑΙΟΥ",
        "ΝΟΜΟΘΕΣΙΑ ΑΝΩΝΥΜΩΝ ΕΤΑΙΡΕΙΩΝ ΤΡΑΠΕΖΩΝ ΚΑΙ ΧΡΗΜΑΤΙΣΤΗΡΙΩΝ",
        "ΠΟΛΙΤΙΚΗ ΑΕΡΟΠΟΡΙΑ",
        "ΕΜΜΕΣΗ ΦΟΡΟΛΟΓΙΑ",
        "ΚΟΙΝΩΝΙΚΕΣ ΑΣΦΑΛΙΣΕΙΣ",
        "ΝΟΜΟΘΕΣΙΑ ΔΗΜΩΝ ΚΑΙ ΚΟΙΝΟΤΗΤΩΝ",
        "ΝΟΜΟΘΕΣΙΑ ΕΠΙΜΕΛΗΤΗΡΙΩΝ ΣΥΝΕΤΑΙΡΙΣΜΩΝ ΚΑΙ ΣΩΜΑΤΕΙΩΝ",
        "ΔΗΜΟΣΙΑ ΕΡΓΑ",
        "ΔΙΟΙΚΗΣΗ ΔΙΚΑΙΟΣΥΝΗΣ",
        "ΑΣΦΑΛΙΣΤΙΚΑ ΤΑΜΕΙΑ",
        "ΕΚΚΛΗΣΙΑΣΤΙΚΗ ΝΟΜΟΘΕΣΙΑ",
        "ΕΚΠΑΙΔΕΥΤΙΚΗ ΝΟΜΟΘΕΣΙΑ",
        "ΔΗΜΟΣΙΟ ΛΟΓΙΣΤΙΚΟ",
        "ΤΕΛΩΝΕΙΑΚΗ ΝΟΜΟΘΕΣΙΑ",
        "ΣΥΓΚΟΙΝΩΝΙΕΣ",
        "ΕΘΝΙΚΗ ΑΜΥΝΑ",
        "ΣΤΡΑΤΟΣ ΞΗΡΑΣ",
```

Image 12: Label names for the volume dataset

5. LABEL AUGMENTATION

In this chapter we discuss the label augmentation technique that we will be using when creating our last, primary model. This technique has initially been introduced in the paper of Chalkidis et al 2020 [1]. The label augmentation technique bases its concept on the assumption that when a label L is assigned to a document, then all of its ancestors should also be assigned to the document. This is intuitively a more fair representation of a hierarchical classification task, but can be quite hard to implement on practice. The key is that in the augmented case, assuming that the ancestors are correctly identified the overall score that will be received would be higher than in the case of having wrongly assigned the ancestors.

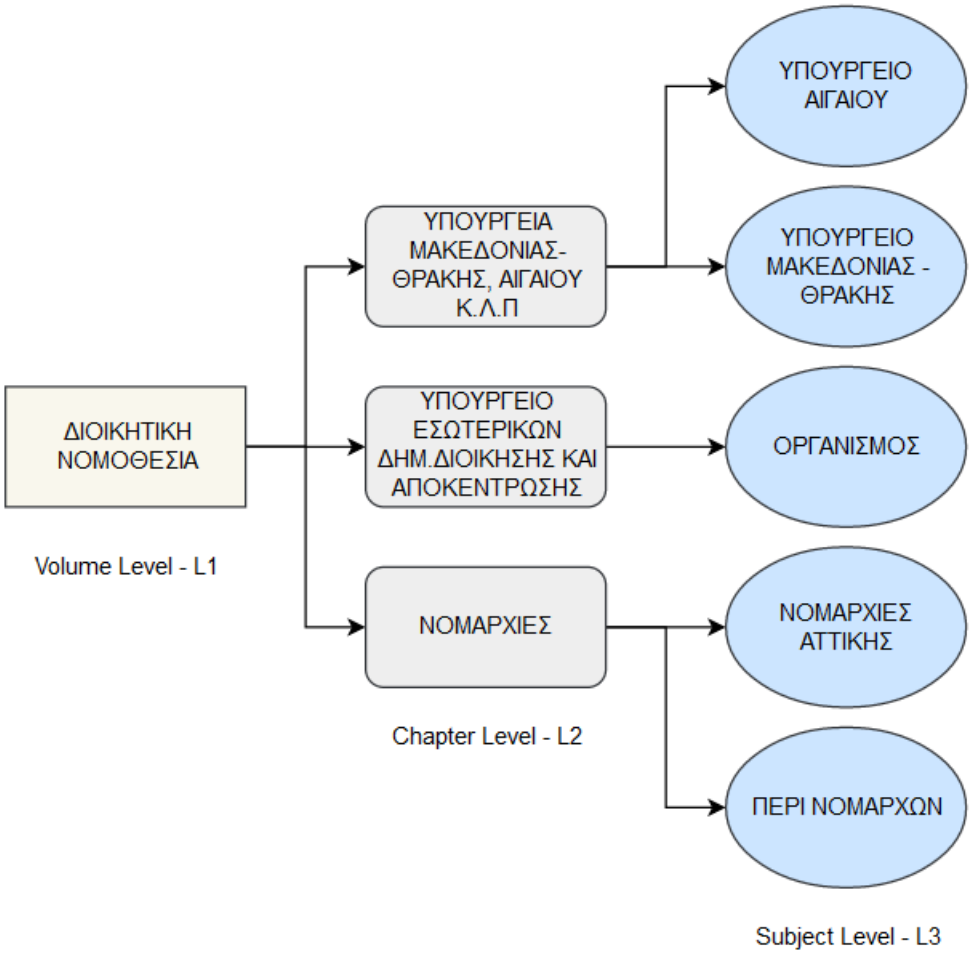

Figure 3: Examples from the GLC hierarchy

For instance, in our case of the GLC dataset, if a document is annotated at the subject level with the label “ΠΛΗΡΩΜΗ ΜΙΣΘΩΝ ΚΑΙ ΗΜΕΡΟΜΙΣΘΙΩΝ” it will also be annotated with its ancestors in both chapter and subject level, that is “ΠΛΗΡΩΜΗ ΕΡΓΑΣΙΑΣ” and “ΕΡΓΑΤΙΚΗ ΝΟΜΟΘΕΣΙΑ” respectively. This assumption is perfectly valid, while also making the computation of predicting the higher levels much more accurate.

A correct assignment of all 3 labels will sum up to a higher score for our model and therefore that will ultimately lead to better predictions. The shallow level of hierarchies, as well as the instance of a lot of few-shot and zero-shot classes in the subject levels make this problem unique and worthwhile to investigate and experiment upon.

The actual way label augmentation works is described as follows: We are given a structured label hierarchy, namely H, of a known depth, namely *d*. H_n is used to denote the set of labels in the n^{th} level. Then, we change our initial labels sets so that a label in a lower hierarchy level, for example the subject level, will also contain the upper levels' ancestors for that label, namely the chapter level and the volume level in this example. Intuitively, this technique will lead to better results for our model, since it will narrow down the possible label candidates in a much more cohesive manner by doing a sort of grouping of the lower level labels with their ancestors.

We can easily find which labels a specific document is assigned with, just by referencing its position on each of our datasets. Then, we can also inspect the actual label names from the pre-loaded variable the GLC provides.

```
pprint(sbj_train_labels[1])
pprint(cht_train_labels[1])
pprint(vol_train_labels[1])
```

```
555
183
35
```

```
pprint(_LABEL_NAMES['volume'][35])
pprint(_LABEL_NAMES['chapter'][183])
pprint(_LABEL_NAMES['subject'][555])
```

```
'ΔΙΟΙΚΗΤΙΚΗ ΝΟΜΟΘΕΣΙΑ'
'ΥΠΟΥΡΓΕΙΟ ΕΣΩΤΕΡΙΚΩΝ ΔΗΜ.ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΑΠΟΚΕΝΤΡΩΣΗΣ'
'ΟΡΓΑΝΙΣΜΟΣ'
```

Image 13: Code that shows how we can access each label level and their names

This label augmentation technique also comes with a modification of the BERT model. For the prediction, the corresponding [CLS] token of a chosen BERT layer is used. This means that different layers can be used to predict different hierarchy levels. Since we have just 3 hierarchy levels, our model will be using the LAST-THREE method of predicting. This method essentially uses the classifiers of the 3 last layers of the model to make predictions. This is called layer-wise guided training and it provides the benefit of having increased accuracy when paired with the label augmentation mechanism.

6. FINE TUNING THE MODELS

In this section we will describe in detail the methods that outline the development processes, not only mentioning the pre-processing, training and evaluation decisions but also presenting the inner workings of each model, as well as the evaluation results that derived from the evaluation. We make sure to take advantage of the best practices and state-of-the-art methods in our development processes. The way this section is presented aims to make the understanding of those development processes as simple as possible while making sure to not miss out on any details that lead to the end results.

In accordance, we start by introducing the pre-processing techniques that were used for the dataset and then following up with the batch definition, the hyper-parameters selection, the optimizer, the cost function and the weight balancing methods that were used. We finish up with the training and evaluation loops.

After having presented the outline of the development processes, we are ready to scrutinize our three models' details, starting from our simpler model and ending up to the final model, which is the primary model of this thesis. At each of the models' sections, the evaluation results are also presented with some light comments comparing it with the previous ones.

6.1 Pre-processing

The pre-processing part is the same for all the models, and it focuses mostly on the dataset. Firstly, we have to look into how the tokenizing will be implemented. For that purpose, we take advantage of the *transformers* library tokenizer: *BertTokenizerFast.*

A tokenizer is essentially in charge of preparing the inputs for a model. The Huggingface *tokenizers* library provide base classes for both a full implementation namely *PreTrainedTokenizer* and a fast one, *PreTrainedTokenizerFast which we will be using*. The fast one is known to achieve significant speed-up improvements and also provide some additional mapping methods. *BertTokenizerFast* is based on WordPiece tokenizer and essentially inherits from *PreTrainedTokenizerFast* which contains most of the main methods.

We can easily load the *BertTokenizerFast* like so:

```python
# Load the GREEK-BERT tokenizer
tokenizer = BertTokenizerFast.from_pretrained('nlpaueb/bert-base-greek-uncased-v1')
```

Python

Image 14: Tokenizer initialization from the pretrained GreekBERT model

taking advantage of the predefined tokenizer of the GreekBERT model.

With this tokenizer as a base, we can define our pre-processing function which will take the 'text' part of each dataset and it will tokenize it with padding and truncation up to a maximum length of 512 tokens. When this function is called, the tokenizer is defined so that it automatically returns the '*input_ids*', the '*token_type_ids*' and the '*attention_mask*'

which are required as input while training the models. The best part is that the tokenizer can also be used for batches without any change.

Keeping a total of 512 tokens is not actually making our predictions worse, because of the actual number of words per set. As shown at Chapter 4 where we describe the dataset, the average number of words for the train set is 600, for the development set 574 and for the test set 595, so very little information is cast away each time.

An important note is that our Greek Legal Code (GLC) dataset consists of legal documents which in most cases contain categorical numbers and unneeded values, especially at the start. We make sure to pre-process each text, essentially truncating arithmetical values and two letter tokens. The tokenizer will then take care of the rest, keeping the most valuable information.

```python
def preprocess_function(examples):
    return tokenizer(examples['text'], padding=True, truncation=True, max_length=512)
```

Image 15: Pre-processing function build upon the tokenizer

Therefore, we apply the pre-processing function to each dataset, making sure to give the parameter *batched* the value '*True*' in the *map* function. We now have the tokenized version of each dataset.

```python
tokenized_vol = volume_dataset.map(preprocess_function, batched=True)
```

Image 16: Mapping of the pre-processing function used on the volume dataset

6.2 Data Loaders

In this part we describe the *DataLoaders*, which are used to split the dataset into batches, as well as providing a single interface for all the three necessary inputs: the sequence *input_ids*, the *attention_mask* and the *labels.* They are provided by the Pytorch utils library and are of great help when developing models like BERT because they keep everything into one place.

First of all we define the a batch size of 8. We chose this low number for speed up and memory management purposes while testing showed that bigger numbers made training almost impossible to handle. To initialize a *DataLoader*, we need to wrap the three inputs into a *TensorDataset* and initializing a sampler with that. Afterwards, the *DataLoader* can be defined given the above mentioned batch size.

The *DataLoader* is essentially a way to store both the samples and the corresponding labels into a single *Dataset* that enables easy access to the samples. All the above can be summarized in the following lines of code:

```
#define a batch size
batch_size = 8

# wrap tensors
train_data = TensorDataset(train_seq, train_mask, train_y)

# sampler for sampling the data during training
train_sampler = RandomSampler(train_data)

# dataLoader for train set
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=batch_size)
```

Image 17: DataLoader initialization for the train set

Of course, we use the same processing for the validation and test data from each hierarchy level; volume, chapter and subject datasets. These *DataLoaders* will be used later on inside the training and evaluation loops.

6.3 Optimizer & Cost Function

A very important decision is the choice of both the optimizer and the cost function. For the former, AdamW seems the most logical choice [35], for the later, we decide to use the classic Cross Entropy Loss function [36]. We will take a small look into those before moving on with the actual training part.

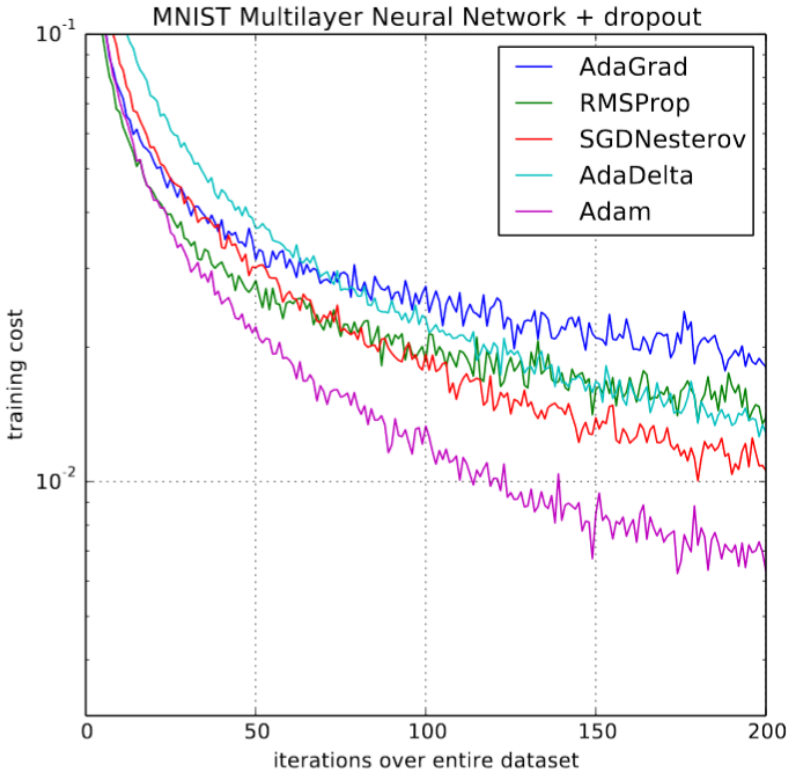

Figure 4: Comparison between Adam and other optimizers

As far as the AdamW optimizer is concerned, it has been shown to not only yield super fast training speeds like Adam, but also to improve it by decoupling the weight decay from the optimization step. This means that the optimal weight decay is not affected by

changes to the learning rate, and has been proven to be the state-of-the-art option in many models. It's the best adaptive optimizer in most of the cases and it is good with sparse data. Essentially, by using AdamW there is no much need to focus on the learning rate value and we will not be needing heavy fine-tuning.

The decision to use Cross Entropy Loss is pretty self explanatory; it is the go-to option when dealing with multi-label classification tasks. In actuality, we will be using the so-called Sigmoid Cross-Entropy loss, which basically is a Sigmoid activation function, plus a Cross Entropy Loss function. To handle cases of class imbalance, we use the weighted version, by first computing the class weights using a function provided by *sklearn* namely *compute_class_weight.* We then convert the weights into a tensor, which may be passed as argument to the corresponding PyTorch *CrossEntropyLoss* function.

6.4 Training and Evaluation

Before going into the models, we will be presenting the training and evaluation loops, and by doing so we will have completed the outline of our development process. It may seem simple at first, but the actual details are crucial to the making of a robust model. For training, we have to make sure of clearing and calculating the gradients at the correct time in each pass while also being wary of the exploding gradients problem that may occur. The evaluation loop is a simpler version of the training counterpart.

First of all, we will be using Huggingface Transformers along with PyTorch for our training and fine-tuning later on. They have been designed to be compatible and abstract a lot of the complexity of the training part specifically.

```
# function to train the model
def train():

  # put the model in training mode
  model.train()

  # iterate over batches
  for step, batch in enumerate(train_dataloader):

    # load the input from the dataloader
    sent_id, mask, labels = batch

    # clear previously calculated gradients
    model.zero_grad()

    # get model predictions for the current batch
    output = model(sent_id, mask)
    preds = output[0]
```

Image 18: Base code for the training loop of our models

We start by calling the method *model.train()* to put the model in train mode. Then we define the iteration over the batches with the help of our *DataLoader*, at each iteration start we have to clear the previously calculated gradients with *model.zero_grad()*. After that we get our model predictions for our current batch and we compute the loss between the actual and the predicted values, summing it to the total loss.

We then use our loss function's *backward()* function call to make a backward pass and calculate the gradients. A very important thing to note here is that we have to prevent the problem of exploding gradients. We can do so by a simple call to *clip_grad_norm_()* function provided by PyTorch. Without this call, we risk losing a lot in prediction accuracy.

```
# compute the loss between actual and predicted values
loss = cross_entropy(preds, labels)

# backward pass to calculate the gradients
loss.backward()

# clip the the gradients to 1.0. It helps in preventing the exploding gradient problem
torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

# update parameters
optimizer.step()
```

Image 19: Continuation of the training loop code

Lastly, we update our optimizer parameters by using the *step()* function. Our function returns the training loss of the epoch as well as the total predictions. These values will be used after the training loop to compare the losses.

Now, for the evaluation loop we have to change some things. First and foremost, we deactivate the dropout layers by calling *model.eval()*, which puts the model in evaluation mode. This time, we do not have to make any changes to our optimizer or loss function, however we have to make sure that the predictions are made with *torch.no_grad()* which deactivates the *autograd* of our model. This is important because we are doing evaluation and not training; we do not want our model's parameters to be updated.

After that, along with the loss, we compute our metrics: R-Precision, Recall and F1-Score and return them. These will be used later on for our models comparison.

6.5 Summary

In this chapter we presented the outline of the development process that will be the main core of our models. Both the training and evaluation loops were described in detail as well as the decision making in our hyper-parameters, cost function, optimizer and DataLoaders. We are now in position to present our models architecture and evaluate them in the next chapter.

7. MODELS & COMPARISON

Having defined the outline of our training and evaluation methods, we can now proceed unto the actual model development. What we aim to create is a model that utilizes the label augmentation technique, on our legal documents dataset and classifies those documents as accurately as possible, given our resources.

This model has its perks against other mainstream models that can be used since it will be different in the way it handles the output predictions and calculates the loss. Nevertheless, the core processes remain the same and little has to be changed outside the models' structure.

That being said, we have to have models that may be used for comparison purposes and these are presented firstly, from the simplest model up to the more complex one. All of the models used follow the training and evaluation methods described above and base their architecture to GreekBERT and essentially BERT. That means that we can utilize predefined functions from HuggingFace and PyTorch that are build for BERT-like models which will help speed-up development time.

Ultimately, we gather each models' metrics into a corresponding table and compare them. The datasets' zero-shot and few-shot labels prove to be a significant obstacle in our models' performance but since each model is trained for the same amount of epochs, 10 in total, the comparison can prove which model is the best.

7.1 Vanilla Model

The vanilla model is our simplest of the three models. It uses the GreekBERT model as its base with the only change of having an increased number of labels in the configuration depending on the hierarchy level. Afterwards, we pass that new configuration to a predefined class provided by HuggingFace, called *AutoModelForSequenceClassification*.

What this does is essentially taking advantage of the *AutoModel* interface which directly chooses the architecture that we want, namely a *BertModel* but this simplifies things since we only want to change the configuration so that we predict more labels. The code for this model is provided below.

```
config = BertConfig.from_pretrained("nlpaueb/bert-base-greek-uncased-v1")
config.num_labels = 47
config.return_dict = False
model = AutoModelForSequenceClassification.from_config(config)
```

Image 20: Code for the vanilla volume case model

Here we can directly appreciate the simplicity and easiness of use that the pretrained transformer models have; in just 4 lines of code we have loaded the model and have created a base model that will be used for sequence classification for the exact number

of labels we want. The code for the chapter and volume cases is the same, except of the number of labels.

However, we must also consider the need of further fine-tuning this model, which will be done in the next model. Without our specialized fine-tuning, the model might incorrectly use a sub-optimal layer in predicting the output label.

Essentially, we will be training three vanilla models, each with different number of labels and on the corresponding labels' dataset. So for our fist vanilla model we predict 47 labels, for our second vanilla model 389 labels and for the third vanilla model 2285.

The optimizer that we will be using is the AdamW provided as well from the transformers library. For the loss function, the cross entropy loss function is used, which is popular for multi-classification problems as mentioned previously.

The models are trained in 10 epochs in total, and evaluated after each pass on the development dataset. The weights of the best model over all epochs are saved in a file, for each model correspondingly. Basically, they follow the training and evaluation processes that were described in *Chapter 6*.

A very important thing to note is that the models do not really change the inner structure of our base model, which means that in the case of our classification task we essentially are not using any fine-tuning for classification. For that reason we can expect a somewhat low level of overall performance.

Table 3: R-Precision ± std for each model, using the optimal saved weights

R-Precision	Training Set	Evaluation Set	Test Set
Volume Level Model	69.5 ± 0.2	68.4 ± 0.1	67.7 ± 0.3
Chapter Level Model	67.5 ± 0.3	68.2 ± 0.3	65.8 ± 0.2
Subject Level Model	66.7 ± 0.2	66.3 ± 0.1	65.3 ± 0.3

As we can see, the models decrease in performance as we go down the hierarchy levels. This is to be expected since the amount of few-shot and zero-shot classes when going down in the hierarchy also increases, and as a result the models learn much slower and less accurately. The Recall and F1-Score metrics results are quite similar to R-Precision.

Table 4: Recall ± std for each model, using the optimal saved weights

Recall	Training Set	Evaluation Set	Test Set
Volume Level Model	69.3 ± 0.4	68.2 ± 0.2	67.5 ± 0.1
Chapter Level Model	67.2 ± 0.3	68.4 ± 0.1	65.6 ± 0.2
Subject Level Model	66.5 ± 0.3	66.5 ± 0.0	65.7 ± 0.4

Table 5: F1-Score ± std for each model, using the optimal saved weights

F1 Score	Training Set	Evaluation Set	Test Set
Volume Level Model	69.4 ± 0.3	68.3 ± 0.2	67.6 ± 0.2
Chapter Level Model	67.3 ± 0.3	68.3 ± 0.2	65.7 ± 0.2
Subject Level Model	66.6 ± 0.3	66.4 ± 0.1	65.5 ± 0.4

Now that we have finished our vanilla model that we can use to compare to our primary model, we need a more advanced version of it. The reason we build this vanilla model is to have an idea of what the lowest possible performance for our models should look like. This way, we may experiment with different hyper-parameters and fine-tuning approaches and easily dismiss those that yield lower or exact same performance. We can now move on with the presentation of our fine-tuned model.

7.2 Fine-tuned model

The second, fine-tuned model is essentially the vanilla model with an extra layer that fine-tunes the output so that the performance is better. To reduce implementation time, the model is predicting labels from all 3 hierarchy levels at the same time, instead of having three separate models for each level.

What that extra layer does is essentially ignoring the pooler output layer of the GreekBERT model, which is known for having decreased performance in text classification tasks. We must also note that the BERT models used have not been directly designed to tackle the task of text classification, making this small adjustment necessary.

After experimenting with the hyper-parameters, we found that by starting with an even lower learning rate yields better results compared to our previous models while other changes do not seem to offer an advantage and a lot of the times its actually a loss in performance. Therefore, the extra layer is our only essential difference to the vanilla model.

Table 6: R-Precision ± std for each hierarchy level, using the optimal saved weights

R-Precision	Training Set	Evaluation Set	Test Set
Volume	73.4 ± 0.3	72.6 ± 0.2	72.7 ± 0.1
Chapter	72.1 ± 0.1	75.0 ± 0.2	73.4 ± 0.3
Subject	69.8 ± 0.3	70.3 ± 0.1	68.2 ± 0.3

By looking at the R-Precision of our fine-tuned model we observe a 5% improvement in predictions, which is very important. The pooler output layer essentially made predictions more obscure in our vanilla model. The smaller learning rate should also be noted since our experiments show an increase of 1-2% in performance which is definitely notable.

Table 7: Recall ± std for each hierarchy level, using the optimal saved weights

Recall	Training Set	Evaluation Set	Test Set
Volume	71.3 ± 0.1	70.5 ± 0.2	70.8 ± 0.3
Chapter	72.8 ± 0.2	75.3 ± 0.1	74.4 ± 0.1
Subject	69.4 ± 0.2	69.3 ± 0.3	67.5 ± 0.2

Table 8: F1-Score ± std for each hierarchy level, using the optimal saved weights

F1 Score	Training Set	Evaluation Set	Test Set
Volume	72.3 ± 0.2	71.5 ± 0.2	71.7 ± 0.2
Chapter	72.4 ± 0.2	75.1 ± 0.2	73.9 ± 0.2
Subject	69.6 ± 0.3	69.8 ± 0.2	67.8 ± 0.3

The Recall and F1-Score metrics follow the trends of R-Precision. We expect training for more than just 10 epochs to further enhance these results but our computation time and overall cost would be much higher. Our goal basically is to compare different techniques so the improvement percentage is what we want to be paying attention to. The fine-tuned model will be a more powerful comparator.

A small note can be made here about how the performance of the model for the chapter level increases in the evaluation set, but we believe that is because of the number of more predictable documents rather than overfitting. Moreover, the predictions on the test set are a bit better than our training set but this can be attributed to the randomness of document allocation in each set.

To sum up, compared with our previous vanilla model, we notice a small but significant improvement in all of our metrics. Of course, the decreasing performance while going down in the hierarchy levels still persists, just like in the case of the vanilla model, which is to be expected due to the increasing amount of few-shot and zero-shot labels of the dataset. We are now ready to move to our primary model.

7.3 Label Augmented model

Our last model is designed to utilize the label augmentation technique. On top of that it's designed so that specific layers are used to predict labels from specific hierarchy levels.

This model is also based on GreekBERT, similarly to our two previous models but it consists of a much more complex architecture.

In order to enable our model to output hidden states, we set the *output_hidden_states* variable in the configuration to True. This also helps with accuracy, since more hidden states can give better accuracy than just one last hidden state which is the default. There are 12 hidden states ion total, corresponding to all the models' layers, from beginning to the last. Each hidden state is essentially an array of shape *(batch_size, sequence_length, hidden_size)* and with this, we can access any of the 12 hidden states.

Basically, for the actual technique used we take advantage of the last three layers of the model for our predictions. We call this technique LAST-THREE, because the classifiers f9-f12 are used to predict the labels in L1 through L3.

```python
config = BertConfig.from_pretrained("nlpaueb/bert-base-greek-uncased-v1")
config.output_hidden_states = True
model = AutoModelForSequenceClassification.from_config(config)
```

Python

Image 21: Initializer code for the pretrained GreekBERT model

The LAST-THREE technique works better in our case compared to other techniques available for label augmented models, since the hierarchy is shallow. This means that while layers 1-8 retain and enhance their pre-trained representations, the last layers, 9-12 will leverage all this previously acquired knowledge to make even better predictions, and it's intuitively the highest level of contextualization possible.

These last layers are by design better at classifying our sequences, since they are forced to handle the classification of gradually more refined classification tasks.

```python
class LabelAugmentedModel(nn.Module):
    def __init__(self):
        super(LabelAugmentedModel, self).__init__()
        self.config = BertConfig.from_pretrained("nlpaueb/bert-base-greek-uncased-v
        self.config.output_hidden_states = True
        self.greek_bert = AutoModelForSequenceClassification.from_config(self.confi
        self.volume = nn.Linear(768, 47)
        self.chapter = nn.Linear(768, 389)
        self.subject = nn.Linear(768, 2285)

    def forward(self, ids, mask):
        outputs = self.greek_bert(ids, attention_mask=mask)
        hidden_states = outputs[2][1:]
        # we can now access the cls token of any hidden state we want
        last_one = hidden_states[2][1]
        last_two = hidden_states[2][2]
        last_three = hidden_states[2][3]

        # We make predictions for each level using different hidden states/layers
        pred_vol = self.volume(last_three.view(-1, 768))
        pred_chapt = self.chapter(last_two.view(-1, 768))
        pred_subj = self.subject(last_one.view(-1, 768))
```

Image 22: Code showing how the hidden states are utilized inside the model for predictions

From our preliminary experiments we found that using a drop-out layer is actually detrimental for our predictions and therefore we will not be using one. As for the hyper-parameters of our model, we leverage the grid-search function to find the optimal learning rate.

As mentioned in our optimizer selection segment of Chapter 6, by using the AdamW optimizer we can be less worried about the learning rate values we try, so we can speed up searching time by looking at just a few learning rates, namely [2e-5, 4e-5, 2e-6].

The training method is very similar to our previous models, but this time we have to make adjustments in both the cost function and the weight balancing.

For the weight balancing, we use the same functionality presented in the models of Chalkidis et al., 2020 [1] where each loss is essentially weighted by the percentage of labels at the corresponding level. This fine-tuning is intuitively correct since different levels have different total amount of labels to predict each time and the weights must be changed according to their number. To put it in mathematical terms, if $|L_n|$ is the number of layers in the nth level of the hierarchy and $|L|$ denotes the total number of labels across all levels, then the weight balancing is $w_n = \frac{|L_n|}{|L|}$ for the nth level. This adjustment is necessary and without it our model will have very low performance scores.

As far as the loss function is concerned, we use the already described loss function for the label augmented model. In detail, since we are using the LAST-THREE technique and our hierarchy is of 3 levels, we need to only adjust the scoring for these last three levels. Our model is graded with a higher value for each correct prediction in each level. This way, our model is fairly evaluated with respect to the augmented labels case. This adjustment is also necessary for our model.

Lastly, we define our classification function $f_i = \sigma(W_i \cdot c_i + b_i)$ which is also in-line with the corresponding classification function used in the model of Chalkidis et al. 2020. [1] Here is a brief description of the parameters:

- L_n is the set of labels in the nth level of our hierarchy, just like in the weight balancing formula.
- W_i is a label vector of the ith layer of our model, and $W_i \in \mathbb{R}^{|L_n| \times 768}$. Here, L_n is changed according to the exact number of labels our layer will predict. This is a trainable parameter for our model.
- b_i is another trainable vector of the ith layer in our model, and $b_i \in \mathbb{R}^{|L| \times 1}$.
- c_i is the classification, [CLS] token of our model. Just like in the case of our fine-tuned model, we make sure to ignore the pooler output layer which obscured our prediction results.
- σ is a Sigmoid activation function, provided by PyTorch.

By using this classification function for training we can be better at training our model. Of course, our pre-trained parameters will have to be utilized as well, so we use these as a starting ground for our classification function parameters by directly extracting them with the *model.parameters()* method.

The label augmented model takes longer to train, due to the redefined cost function, the modified per-layer weight balancing as well as our new classification function, however the predictions should intuitively be more accurate. We present the results below.

Table 9: R-Precision ± std for each hierarchy level, using the optimal saved weights

R-Precision	Training Set	Evaluation Set	Test Set
Volume Level	88.9 ± 0.2	88.9 ± 0.2	88.7 ± 0.1
Chapter Level	87.5 ± 0.3	87.2 ± 0.1	87.4 ± 0.2
Subject Level	86.6 ± 0.2	86.8 ± 0.3	86.7 ± 0.2

Table 10: Recall ± std for each hierarchy level, using the optimal saved weights

Recall	Training Set	Evaluation Set	Test Set
Volume Level	87.9 ± 0.3	87.9 ± 0.3	87.6 ± 0.3
Chapter Level	86.5 ± 0.1	87.1 ± 0.2	86.2 ± 0.3
Subject Level	83.2 ± 0.3	83.4 ± 0.2	83.1 ± 0.1

Table 11: F1 Score ± std for each hierarchy level, using the optimal saved weights

F1 Score	Training Set	Evaluation Set	Test Set
Volume Level	88.4 ± 0.3	88.4 ± 0.3	88.1 ± 0.2
Chapter Level	87.0 ± 0.2	86.6 ± 0.2	86.8 ± 0.3
Subject Level	84.9 ± 0.3	85.1 ± 0.3	84.9 ± 0.2

The results show that our technique really worked in increasing the prediction accuracy. An important thing to note here is that the decrease of the performance going down in hierarchy levels is much more smoother, which shows that our scoring function helped a lot to group together the correct different labels from each level.

7.4 Summary

In this chapter we made a detailed presentation of our three developed models with increased focus on our last and primary model. The evaluation metrics were also presented and a comparison of the aforementioned models was conducted. In our next chapter, we make a more detailed comparison of our models and discuss our findings and final remarks.

8. CONCLUSIONS

In this chapter we will compare in more detail our three models and discuss how our different techniques led to a better performance. The metric of our comparison will be F1 Score. Furthermore, we will present our predictions only on the test set but for each hierarchy level. We can see a very big gap of performance when using the label augmented model in comparison to the previous ones. This improvement is about 21% when compared to the vanilla model and almost 17% compared to the fine-tuned model for the volume level.

Table 12: Comparison table for each developed model

F1 Scores on Test Set	Vanilla Model	Fine-Tuned Model	Label Augmented Model
Volume Labels 47	67.6 ± 0.2	71.7 ± 0.2	88.1 ± 0.2
Chapter Labels 389	65.7 ± 0.2	73.9 ± 0.2	86.8 ± 0.3
Subject Labels 2285	65.5 ± 0.4	67.8 ± 0.3	84.9 ± 0.2

Going down in the hierarchy, we observe a 19% improvement compared to the vanilla model and a 13% for the fine-tuned model for the chapter level. Lastly, for the subject level we see a 19% improvement compared to the vanilla model and a 17% for the fine-tuned model. This constant enhance in performance shows how much important is to group as much as possible the labels from different hierarchy levels in hierarchical classification and further enhances the point of using label augmentation when dealing with hierarchical content while also showing that the better design of the task leads to better results.

We also believe that the layer-wise guided training is essential to our models performance and that without it the prediction accuracy would be much lower. This is again accounted to our different way of dealing with the classification task. Training the last three layers for predictions while making lower layers deal with lower hierarchy levels is intuitively a better training approach to using only one instance of the last layer to predict all three labels.

We must also take into consideration that the label set for both chapter and subject levels is quite large, which would intuitively introduce a high level of difficulty and loss in performance when using label augmentation. In practice, the computational time is approximately the same in all of our models which is of course supported by parallelization techniques. That being said, our computational cost is higher in our last case, essentially because we increase the amount of predicted labels and make the cost function a bit more complex.

Consequentially, we have produced three different models for handling the task of hierarchical multi-label text classification, one of which utilizes the label augmentation mechanism. This model produces significantly better results and further enhances the argument of using label augmentation in any kind of hierarchical text classification problem.

Despite its increased performance, models that use the technique should always be aware of what layers will be utilized in the prediction. In our case, LAST-THREE was the best option due to the shallow depth of the hierarchy structure and was our direct choice given the experiments presented in the paper of Chalkidis et al. 2020. [1] In other problems, different amount and kind of layers will probably have to be used and experimenting with the possible combinations will be needed.

A final note should be made on the intricacies of the GLC dataset. The last two hierarchy levels contain a lot of few-shot and zero-shot classes, making it more difficult for the label augmentation technique. Nevertheless, the model has no problem correctly classifying the sequences into every hierarchy level since it takes advantage of basically grouping together the correct labels by giving them higher scores. Having such a large dataset of legal documents can really obscure the predictions and make the training part really difficult, so correct handing of our input is absolutely necessary, as presented in the pre-processing section.

To summarize, label augmenting paired with label-wise guided training leads to enhanced results even in the case of a large set of label-ancestors in a shallow hierarchical structure and it should be a considerable choice in multi-label classification models. Of course, different datasets would need different handling and result may vary.

ABBREVIATIONS – ACRONYMS

NLP	Natural Language Processing
NER	Named Entity Recognition
POS	Part Of Speech Tagging
GLC	Greek Legal Code
Seq2Seq	Sequence-To-Sequence
RNN	Recurrent Neural Network
LSTM	Long Shot Term Memory
GRU	Gated Recurrent Unit
BERT	Bidirectional Encoder Representations from Transformers
CBOW	Continuous Bag Of Words

REFERENCES

[1] Nikolaos Manginas, Ilias Chalkidis and Prodromos Malakasiotis. Layer-wise Guided Training for BERT: Learning Incrementally Refined Document Representations. arXiv:2010.05763 [cs.CL]

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL]

[3] John Koutsikakis, Ilias Chalkidis, Prodromos Malakasiotis and Ion Androutsopoulos. GreekBERT: The Greeks visiting Sesame Street. arXiv:2008.12014 [cs.CL]

[4] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest and Alexander M. Rush. HuggingFace's Transformers: State-of-the-art Natural Language Processing. arXiv:1910.03771 [cs.CL]

[5] Christos Papaloukas, Ilias Chalkidis, Konstantinos Athinaios, Despina-Athanasia Pantazi and Manolis Koubarakis. Multi-granular Legal Topic Classification on Greek Legislation. arXiv:2109.15298 [cs.CL]

[6] Iulia Turc, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. Well-Read Students Learn Better: On the Importance of Pre-training Compact Models. arXiv:1908.08962 [cs.CL]

[7] Jacob Devlin and Ming-Wei Chang. Research Scientists, Google AI Language. Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing. Google AI Blog

[8] Marwan Omar, Soohyeon Choi, DaeHun Nyang and David Mohaisen. Robust Natural Language Processing: Recent Advances, Challenges, and Future Directions. arXiv:2201.00768 [cs.CL]

[9] Daniel W. Otter, Julian R. Medina and Jugal K. Kalita. A Survey of the Usages of Deep Learning in Natural Language Processing. arXiv:1807.10854 [cs.CL]

[10] Dan Hendrycks, Kimin Lee and Mantas Mazeika. Using Pre-Training Can Improve Model Robustness and Uncertainty. arXiv:1901.09960v5 [cs.LG]

[11] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji-Rong Wen, Jinhui Yuan, Wayne Xin Zhao and Jun Zhu. Pre-Trained Models: Past, Present and Future. arXiv:2106.07139v3 [cs.AI]

[12] John P. Lalor, Hao Wu and Hong Yu. Improving Machine Learning Ability with Fine-Tuning. ArXiv

[13] John P. Lalor, Hao Wu and Hong Yu. Soft Label Memorization-Generalization for Natural Language Inference. arXiv:1702.08563v3 [cs.CL]

[14] Ilya Sutskever, Oriol Vinyals and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. arXiv:1409.3215v3 [cs.CL]

[15] Rumelhart, David E. Hinton, Geoffrey E. Williams and Ronald J. Learning internal representations by error propagation. Tech. rep. ICS 8504. San Diego, California: Institute for Cognitive Science, University of California.

[16] Jordan and Michael I.. Serial order: a parallel distributed processing approach. Tech. rep. ICS 8604. San Diego, California: Institute for Cognitive Science, University of California.

[17] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks. arXiv:1909.09586 [cs.NE]

[18] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. arXiv:1412.3555 [cs.NE]

[19] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv:1406.1078v3 [cs.CL]

[20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. Attention Is All You Need. arXiv:1706.03762v5 [cs.CL]

[21] Alex Sherstinsky. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. arXiv:1808.03314v9 [cs.LG]

[22] Lulu Wan, George Papageorgiou, Michael Seddon and Mirko Bernardoni. Long-length Legal Document Classification. arXiv:1912.06905v1 [cs.CL]

[23] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura E. Barnes and Donald E. Brown. Text Classification Algorithms: A Survey. arXiv:1904.08067v5 [cs.LG]

[24] Rie Johnson and Tong Zhang. Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. arXiv:1412.1058v2 [cs.CL]

[25] Takeru Miyato, Andrew M. Dai and Ian Goodfellow. Adversarial Training Methods for Semi-Supervised Text Classification. arXiv:1605.07725v4 [stat.ML]

[26] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola and Eduard Hovy. Hierarchical Attention Networks for Document Classification. https://aclanthology.org/N16-1174

[27] Himanshu S. Bhatt, Manjira Sinha and Shourya Roy. Cross-domain Text Classification with Multiple Domains and Disparate Label Sets. https://aclanthology.org/P16-1155

[28] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu and Pavel Kuksa. Natural Language Processing (almost) from Scratch. arXiv:1103.0398v1 [cs.LG]
[29] Yoon Kim. Convolutional Neural Networks for Sentence Classification. arXiv:1408.5882v2 [cs.CL]
[30] Christopher Olah. Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/
[31] Collins and Singer. Unsupervised Models for Named Entity Classification. EMNLP 1999
[32] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats and Yann N. Dauphin. Convolutional Sequence to Sequence Learning. arXiv:1705.03122v3 [cs.CL]
[33] Andrea Galassi, Marco Lippi and Paolo Torroni. Attention in Natural Language Processing. arXiv:1902.02181v4 [cs.CL]
[34] Sneha Chaudhari, Varun Mithal, Gungor Polatkan and Rohan Ramanath. An Attentive Survey of Attention Models. arXiv:1904.02874v3 [cs.LG]
[35] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. arXiv:1412.6980v9 [cs.LG]
[36] Aditya K. Menon, Ankit Singh Rawat, Sashank Reddi and Sanjiv Kumar. Multilabel reductions: what is my loss optimising?, Advances in Neural Information Processing Systems 32 (NeurIPS 2019)
[37] Jesse Read and Fernando Perez-Cruz. Deep Learning for Multi-label Classification. arXiv:1502.05988v1 [cs.LG]
[38] Konstantinos I. Athinaios. Named Entity Recognition using a Novel Linguistic Model for Greek Legal Corpora based on BERT model, BSc THESIS
[39] Yi Tay, Mostafa Dehghani, Jai Gupta, Dara Bahri, Vamsi Aribandi, Zhen Qin and Donald Metzler. Are Pre-trained Convolutions Better than Pre-trained Transformers?. arXiv:2105.03322v2 [cs.CL]
[40] Kevin Clark, Urvashi Khandelwal, Omer Levy and Christopher D. Manning. What Does BERT Look at? An Analysis of BERT's Attention. https://aclanthology.org/W19-4828/
[41] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong and Qing He. A Comprehensive Survey on Transfer Learning. arXiv:1911.02685v3 [cs.LG]
[42] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781v3 [cs.CL]
[43] Oriol Vinyals and Quoc Le. A Neural Conversational Model. arXiv:1506.05869v3 [cs.CL]
[44] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee and Luke Zettlemoyer. Deep contextualized word representations. arXiv:1802.05365v2 [cs.CL]