NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

BSc THESIS

Language Models for Ancient Greek

Andreas I. Spanopoulos

Supervisor: **Manolis Koubarakis,** Professor

ATHENS

MARCH 2022

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Γλωσσικά μοντέλα για τα Αρχαία Ελληνικά

Ανδρέας Η. Σπανόπουλος

Επιβλέπων: **Μανόλης Κουμπαράκης,** Καθηγητής

ΑΘΗΝΑ

ΜΆΡΤΙΟΣ 2022

BSc THESIS

Language Models for Ancient Greek

Andreas I. Spanopoulos

S.N.: 1115201700146

SUPERVISOR: **Manolis Koubarakis,** Professor

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Γλωσσικά μοντέλα για τα Αρχαία Ελληνικά

Ανδρέας Η. Σπανόπουλος
Α.Μ.: 1115201700146

ΕΠΙΒΛΕΠΩΝ: **Μανόλης Κουμπαράκης,** Καθηγητής

ABSTRACT

BERT is a pre-trained Language Model introduced by Google AI Language in 2018, that manages to achieve state-of-the-art results on many downstream tasks. It is a revolutionary model that can be used to tackle almost any NLP task. It has also been successfully applied to other languages apart from English, such as French, Spanish and even Greek and Latin.

The goal of this thesis was to create a Language Model for the Ancient Greek language based on a BERT-like architecture, and then fine-tune it to some downstream task, specifically Part-of-Speech tagging. There were two main steps that needed to be taken in order to develop the model. First, there was the data collection part, and then training a language model with the data acquired.

Regarding the data collection part, a lot of research was done in order to find publicly available sources with plain-text data, but not much was found. In total, the amount of data that we were able to collect was a bit less than 450 MB. This was a major problem, as most BERT-like models had been trained on corpora in the scale of 30-50 GB.

Regarding the training part, the RoBERTa pre-training and model were chosen as a framework for the Ancient Greek Language Model, since it had demonstrated better performance than other variants. The training objective was that of Masked Language Modelling with dynamic masking.

Unexpectedly, the results were two-fold. One the one hand, the Language Model kept underfitting due to the fact that it wasn't seeing enough data. Many ideas were tried such as reducing the model size and tuning the hyperparameters with bayesian optimization, but none yielded good results. On the other hand, when fine-tuning for PoS Tagging, the results were reasonably good, which suggests that the Language Model has learnt important aspects of the Ancient Greek language.

By taking a look at the training curves, we can see that the model is definitely learning something as the loss keeps decreasing, up until a point where it converges. We strongly believe that this underfitting effect is due to the lack of a much larger corpora. If more data is made available in the future, it would be definitely worth trying out again this approach. That's why the code for downloading the data and training a model is made available at `https://github.com/AndrewSpano/BSc-Thesis`.

SUBJECT AREA: Natural Language Processing

KEYWORDS: Artificial Intelligence, Machine Learning, Natural Language Processing, Deep Learning, Transformers, BERT

ΠΕΡΙΛΗΨΗ

Το BERT είναι ένα προ-εκπαιδευμένο Γλωσσικό Μοντέλο το οποίο αναπτύχθηκε από την ομάδα της Google AI Language το 2018, που πετυχαίνει κορυφαίες επιδόσεις σε πολλά προβλήματα Επεξερασίας Φυσικής Γλώσσας. Είναι ένα επαναστατικό μοντέλο που μπορεί να χρησιμοποιηθεί για την επίλυση σχεδόν οποιουδήποτε προβλήματος απατεί διαχείριση φυσικής γλώσσας. Ήδη, έχει εφαρμοσθεί με επιτυχία σε πληθώρα γλωσσών πέρα από τα Αγγλικά, όπως Γαλλικά, Ισπανικά, αλλά ακόμη και σε Ελληνικά και Λατινικά.

Ο στόχος αυτής της πτυχιακής ήταν η δημιουργία ενός Γλωσσικού Μοντέλου για την Αρχαία Ελληνική γλώσσα βασισμένο σε μια αρχιτεκτονική τύπου BERT, και στη συνέχεια η εφαρμογή του σε κάποιο πρόβλημα, συγκεκριμένα στην αναγώριση ετικετών μέρους του λόγου. Υπήρχαν δύο βασικά βήματα που έπρεπε να πραγματοποιηθούν για να αναπτυχθεί το μοντέλο. Αρχικά, υπήρξε το κομμάτι της συλλογής δεδομένων, και στη συνέχεια η εκπαίδευση ενός γλωσσικού μοντέλου με τα δεδομένα που αποκτήθηκαν.

Όσον αφορά το κομμάτι της συλλογής δεδομένων, πραγματοποιήθηκε αρκετή έρευνα προκειμένου να βρεθούν διαθέσιμες στο κοινό πηγές με δεδομένα απλού κειμένου, αλλά δεν βρέθηκαν πολλά. Συνολικά, ο όγκος των δεδομένων που μπορέσαμε να συλλέξουμε ήταν λίγο μικρότερος από 450 MB. Αυτό ήταν ένα σημαντικό πρόβλημα, καθώς τα περισσότερα μοντέλα που βασίζονται σε αρχιτεκτονική BERT είχαν εκπαιδευτεί με όγκους δεδομένων της κλίμακας των 30-50 GB.

Όσον αφορά το κομμάτι της εκπαίδευσης, η προ-εκπαίδευση και το μοντέλο RoBERTa επιλέχθηκαν ως πλαίσιο για το Μοντέλο της Αρχαίας Ελληνικής Γλώσσας, αφού έχουν επιδείξει καλύτερες επιδόσεις από άλλες παραλλαγές του BERT. Ο στόχος εκπαίδευσης ήταν αυτός του Masked Language Modeling με δυναμική μάσκα.

Απροσδόκητα, τα αποτελέσματα είχαν δύο πλευρές. Από την μία πλευρά, το Γλωσσικό Μοντέλο αδυνατούσε να μάθει εις βάθος την Αρχαία Ελληνική Γλώσσα διότι δεν εκπαιδευόταν με αρκετά δεδομένα. Δοκιμάστηκαν πολλές ιδέες, όπως η μείωση του μεγέθους του μοντέλου και ο συντονισμός των υπερπαραμέτρων με μπαεσιανή βελτιστοποίηση, αλλά καμία δεν έδωσε καλά αποτελέσματα. Από την άλλη πλευρά, όταν το εφαρμόσαμε στο πρόβλημα αναγνώρισης ετικετών μέρους του λόγου, τα αποτέσματα είναι αρκετά καλά, κάτι το οποίο συνιστά πως το μοντέλο είχε όντως κατανοήσει ορισμένες πτυχές της Αρχαίας Ελληνικής Γλώσσας.

Ρίχνοντας μια ματιά στις καμπύλες εκπαίδευσης, μπορούμε να δούμε ότι το μοντέλο σίγουρα μαθαίνει κάτι, καθώς η αντικειμενική συνάρτηση συνεχίζει να μειώνεται, μέχρι ένα σημείο στο οποίο συγκλίνει. Πιστεύουμε ότι αυτή η συμπεριφορά οφείλεται στην έλλειψη ενός πολύ μεγαλύτερου σώματος δεδομένων. Εάν διατεθούν περισσότερα δεδομένα στο μέλλον, σίγουρα θα άξιζε να δοκιμαστεί ξανά αυτή η προσέγγιση. Γι' αυτό τον λόγο, ο κώδικας για τη λήψη των δεδομένων και την εκπαίδευση ενός μοντέλου διατίθεται στη ηλεκτρονική διεύθυνση https://github.com/AndrewSpano/BSc-Thesis.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Επεξεργασία Φυσικής Γλώσσας

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Τεχνητή Νοημοσύνη, Μηχανική Μάθηση, Επεξεργασία Φυσικής Γλώσσας, Βαθειά Μάθηση, Μετασχηματιστές, Μοντέλο BERT

In dedication to my parents, who have always supported me. I wouldn’t have accomplished anything if it wasn’t for you. Thank you, from the bottom of my heart.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Professor Manolis Koumbarakis, for the support and the valuable guidance that he offered me throughout this journey. I would also like to thank Professor Ioannis Panagakis for his valuable contribution to its completion, as well as Michalis Nikolaou for the access he gave us to the HPCF Cyclone in Cyprus, which allowed us to do all our experiments using its resources.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This thesis was written as a part of the BSc program of studies at the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens.

1. INTRODUCTION

The fields of Artificial Intelligence (AI) and Machine Learning (ML) have witnessed exponential growth in the 21st century [2]. Many everyday tasks such as person identification, customer assistance, question answering and many more, have been successfully automated thanks to ML algorithms and models. The majority of these tasks are non-trivial to solve and thus require some level of intelligence in order to tackle them. A significant subset of those tasks deals with interactions between computers and human language. This subset falls within the category of Natural Language Processing.

Natural Language Processing (NLP) [20] is a subfield of Linguistics, Computer Science and Artificial Intelligence and an active area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things. It has a wide variety of applications ranging from text summarization to chatbots and question answering models [13, 28, 39].

Most, if not all, of the recent NLP models that achieve state-of-the-art (SOTA) results, have been developed using a technique called Transfer Learning (TL). A high-level description of this method is “applying knowledge attained from solving a specific problem, to a different problem”. In NLP, Transfer Learning is usually achieved with the usage of pre-trained Language Models (LMs), which are models that have been trained on huge corpora (5-150 GB) to learn universal language representations [31]. Examples of such models are ELMo, BERT and GPT [30, 21, 17]. Combining a pre-trained model with a smaller network and then fine-tuning it, is likely to significantly improve the performance on downstream tasks, by reducing overfitting and achieving more generalization [31]. Thus, it can be argued that it is of paramount importance to achieve good results when pre-training Language Models.

The main target of this тhesis is to pre-train a LM for Ancient Greek, and then test out the performance on the downstream task of PoS Tagging, which is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context [6]. The publicly available data which we were able to leverage, was from Perseus, First1kGreek Project, Portal and the Diorisis Ancient Greek Corpus [7, 3, 9, 1]. Unfortunately, all the combined data consisted of approximately 450 MB, which is a few order of magnitude less than was was used for other languages, such as Latin and Modern Greek [16, 25].

For the choice of a pre-trained model architecture, BERT-like architectures are preferred due to the strong performance they have achieved in previous publications, as well as the fact their number of trainable parameters is one order of magnitude less than GPT-like architectures [21, 17], which allows for much faster experimenting. Small adjustments to the overall architecture will be tried during hyperparameter tuning, to see if better results are achieved.

To our knowledge, the only publicly available pre-trained Ancient Greek Language Models has been developed by Brennan Nicholson and Pranyadeep *et al.* [12, 37]. The former is a bit hard to fine-tune on downstream tasks, as it’s a character-level model, and thus the only application that it’s well-suited for is Masked Language Modelling. The latter is a BERT-based model that makes use of the more efficient word embeddings.

Our documentation is mainly comprised of the four chapters that are listed below.

Chapter 2 will build up the knowledge required to understand the structural components of the state-of-the-art Language Models, as well as for some of their most popular variations. We will talk about Machine Learning in General, Multi Layer Perceptrons, vanilla Recurrent Neural Networks, the attention mechanism and eventually we will build up to the transformers architecture and BERT.

In Chapter 3, we will discuss the setup for developing a pre-trained Ancient Greek LM, the data used for it, the preprocessing steps that were taken, as well as some other implementation-specific details.

In Chapter 4 we will take a look at the results, both for the pre-training of the LM, as well as for Part of Speech Tagging. We will also define some metrics that will help us assess the performance of our model.

In Chapter 5 we will elaborate on the results and discuss future work on this subject.

2. BACKGROUND AND RELATED WORK

Recent advances in attention-based architectures for LMs, have shown that transformer-based architectures are far superior [38] than previous Recurrent Neural Network (RNNs) approaches, such as LSTM and GRU [23, 19]. This sudden increase in performance caused a revolution in the field of NLP, as more and more variations of the transformer framework that crashed the previous benchmarks were being published. Nowadays, this architecture has become the default choice for researchers when experimenting with new methods or models. This also holds for industry-level applications, as most companies have now switched to BERT-like or GPT-like models for their NLP tasks.

Regarding NLP for Ancient languages, BERT has already been successfully applied to Ancient Greek, Modern Greek and Latin [37, 25, 16]. The first and the latter achieve pretty decent results. The Greek-BERT achieved SOTA results in many downstream tasks.

The goal of this chapter is present the foundations of BERT-like architectures. We will briefly explain the building blocks of the transformer architectures, the models that were used prior to BERT-like models, and we will conclude the chapter with a small study on some of the most popular variants of BERT.

2.1 Machine Learning

Over the years, there have been many definitions of Machine Learning, by various researchers. One of first definitions, states that:

"*Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed*" - Arthur Samuel, 1959.

This definition is quite simple, and it successfully manages to capture the true essence of this field. Yet, there is a recent definition that seems even more appropriate:

"*A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E*" - Tom Mitchell, 1997

Indeed, the latter portrays perfectly the scope of ML. But how does learning occur, and what is being learnt? The word "learning" by itself is vague, as it can be interpreted in many ways. That's why ML can be distinguished in three different sub-fields, where each one deals with a different kind of learning.

The first sub-field is called **Supervised Learning**, and it deals with problems where the task is to learn a function (mapping) that maps a given input **X**, to a given output **y**. This function, can be any type of function $f : A \to B$, where A and B are any 2 sets. The output **y** can either be a continuous or a discrete value. If it's continuous, we say that we are dealing with a *Regression* problem, and if it's discrete then it's called a *Classification* problem. In this framework, the task *T* is to learn f, the experience *E* is the data $\mathbf{X}, \mathbf{y}$ which is provided by external sources and the performance *P* is some metric (defined by humans) to assess the performance of f.

The second sub-field is called **Unsupervised Learning**, and it deals with finding patterns in given data. As in *Supervised Learning*, in *Unsupervised Learning* we are given some data **X** but without labels **y**, and the main goal is to find structure inside the given data. Some examples of such algorithms could be to Cluster the data, i.e. find groups of data that exhibit similar features, or reducing the dimensionality of data, by checking which features contribute less. In this framework, the task T is to learn the underlying structure of the given data, the experience E is the data **X** which is provided by external sources and the performance P is again some metric defined by humans in order to assess the quality of the structure found.

The third sub-field is **Reinforcement Learning**, and it deals with a more abstract family of problems, where there is an intelligent agent that interacts with an environment, and it's task is to maximize the notion of cumulative reward received from the environment. Therefore, it deals mainly with control and optimization problems. In this framework, the task T is to learn to act intelligently in the given environment in order to maximize the cumulative reward, the experience E is the environment-specifics which are usually pre-defined and the performance P is usually the maximization of the cumulative reward.

Most problems in NLP (including pre-training a LM) fall within *Supervised Learning*. Hence, from now on we will solely focus in this category. A visual representation of ML and its sub-fields and be seen in fig. 2.1.

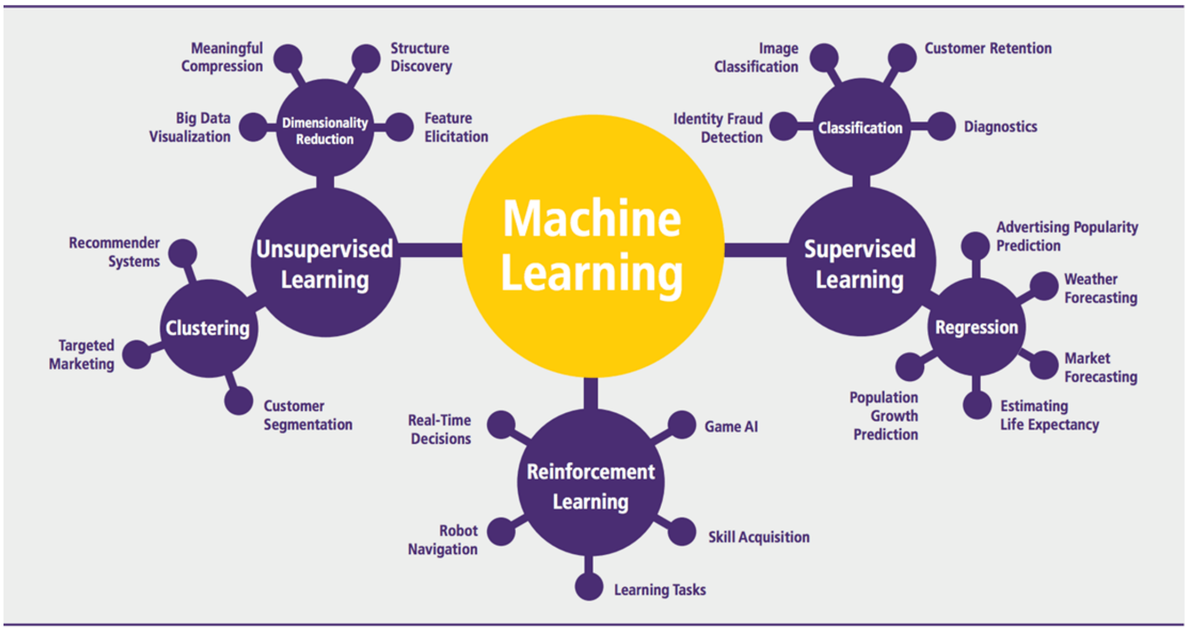


Figure 2.1: Types and examples of Machine Learning algorithms

2.2 Neural Networks

Before diving deep into the theory of ML models, we must present the Artificial Neural Network (ANN), also known as Neural Network (NN). An ANN is a computing system inspired by the biological neural network that constitutes the brain of an animal. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. The connections have a similar functionality to the synapses of a biological brain; they transmit signals between neurons. The "signal" is a real number, and the output of each neuron is computed by some function of the its inputs. These connections are usually referred to as *weights*, and they are adjusted during training. The weight determines the impact of a signal. An increase or a decrease on the weight will determine the strength of the signal. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

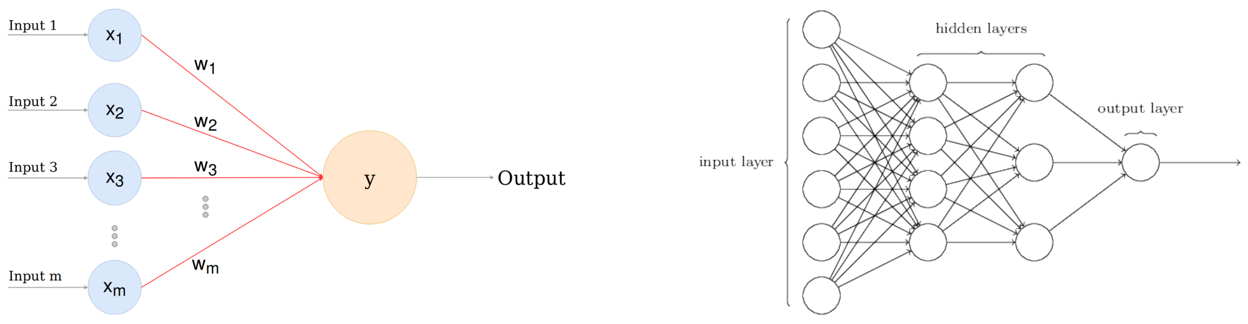
2.3 Perceptron and Multi-Layer Perceptron

The first Neural Network is a binary classifier called “Perceptron” that was introduced by F. Rosenblatt [32] in 1958. It’s the simplest Neural Network; It consists of an input layer, followed by the output layer which is a single neuron. The value of the output neuron can be computed as $f(\mathbf{x}) = \mathbf{wx} + b$, where $\mathbf{w} \in \mathbb{R}^{1\times n}$ is the weight vector (signals), $\mathbf{x} \in \mathbb{R}^{n\times 1}$ is the input vector and $b \in \mathbb{R}$ is the bias. The classifier will then pick a class according to the sign of the output neuron. Namely:

$$\hat{y} = \begin{cases} 0 & \text{if } \mathbf{wx} + b < 0 \\ 1 & \text{if } \mathbf{wx} + b \geq 0 \end{cases}$$

Using the labels y, and the output logits $\hat{y}$, we can assess the performance of the model using some loss function, for example the Negative Log Likelihood (NLL) loss. The weights $\mathbf{w}$ can then be updated by gradient descent. Their gradient with respect to the loss function is computed with the chain rule. The latter process is also known as backpropagation [33].

Of course, this model will not perform well for complicated tasks, since it only learns a linear decision boundary. In order to learn more complex decision boundaries, for classification, or more complex values for regression, we need to increase the size of our model. For this, we can add more layers between the input and output layers, followed by a non-linear activation function in order to learn more complex representations of the input data. These new layers will be called “hidden” layers. These new types of models are called “Multi-Layer Perceptrons” (MLPs). The number of layers to add and their respective size is an architectural choice. Theory has shown that MLPs, also known as Feed Forward Neural Networks (FFNNs), are universal function approximators [24], i.e. by increasing enough their size, then they can approximate any function.



(a) Perceptron (b) Multi-Layer Perceptron

Figure 2.2: A visualization of (a) the Perceptron model and (b) the Multi-Layer Perceptron model

2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs), are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. In other words, they are a generalization of the MLP, that also has an internal memory. Connections between neurons form a directed graph along a temporal sequence, allowing the exhibition of temporal dynamic behavior and the processing of variable-length sequences.

2.4.1 Vanilla RNN

The simplest type of RNN is the one that preserves a single hidden state $a^{<t-1>}$, and uses it when computing the hidden state $a^{<t>}$ for the next input $x^{<t>}$. Mathematically, this can be expressed in the following way: $a^{<t>} = f_W(a^{<t-1>}, x^{<t>})$, where t is the current index of the sequence (aka timestep). The model will need 5 weight matrices in total: 3 Weight matrices to compute $a^{<t>}$ and 2 to compute $\hat{y}^{<t>}$, which is the output of the model at timestep t. The respective equations are:

$$a^{<t>} = \text{tanh}\left(W_{hh}a^{<t-1>} + W_{xh}x^{<t>} + b_h\right)$$
$$\hat{y}^{<t>} = W_{hy}a^{<t>} + b^{<t>}$$

$a^{<0>}$ is usually initialized to be the ones matrix. There are four main types of RNN architectures, where the input and output are modelled in different ways. Each usually targets different kind of problems. Those variations can be seen in fig. 2.3.

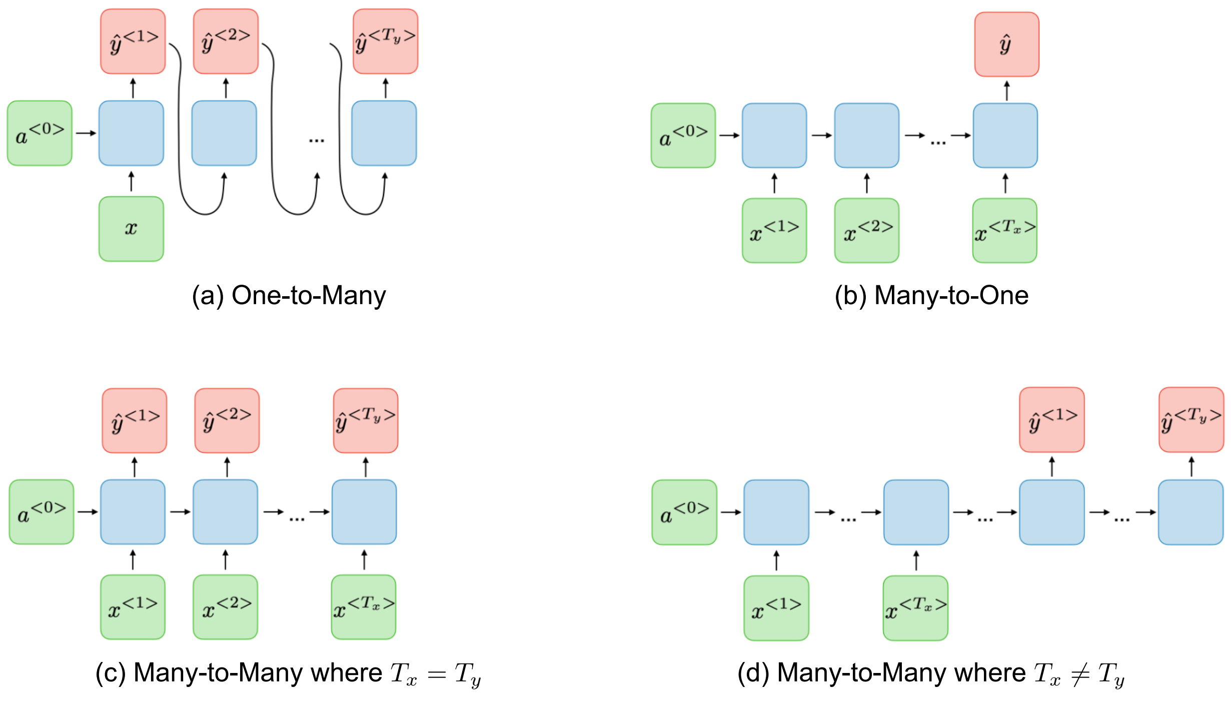


Figure 2.3: Different types of RNN architectures. (a) is usually used in text/music generation. (b) is usually used classification/regression problems. (c) is usually used in Multi-Label Classfication/Regression problems, such as Named Entity Recognition or Part-of-Speech Tagging. (d) is usually used in Machine Translation.

Even though the Vanilla RNN architecture is itself a Universal Function Approximator [35], it suffers from 2 main problems, those of **Exploding Gradients** and **Vanishing Gradients**. The **former** refers to the problem where the gradients computed during backpropagation diverge to very high values, thus rendering them useless. This problem can be easily fixed by clipping the gradients to some threshold upper bound. The **latter** refers to the problem where the gradients of earlier timesteps converge to really small values, thus causing the model to stop learning. This happens because gradients are computed using the chain rule, which is a multiplication of 2 values. If both values are smaller than 1, then the gradients shrink in each timestep, reaching values close to 0 as the backpropagation goes deeper.

The Vanishing Gradients problem is the biggest issue with Vanilla RNN architectures, since this is what stops them from capturing long-term dependencies. This issue is illustrated in fig. 2.4, where we can see that in the sentence “What time is it?”, the contribution of the word “What” to the hidden state of the word “?” is minimal.

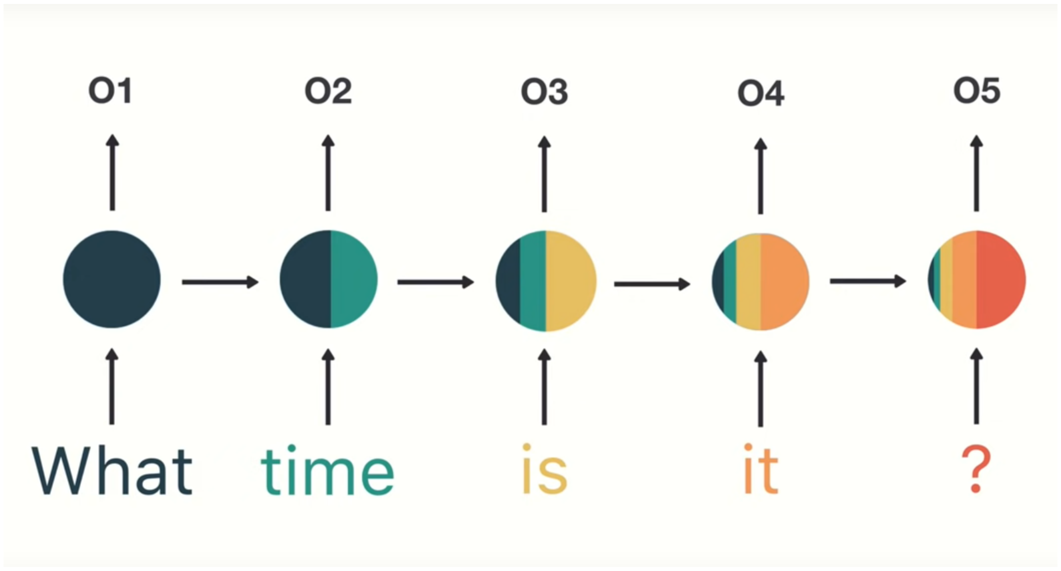


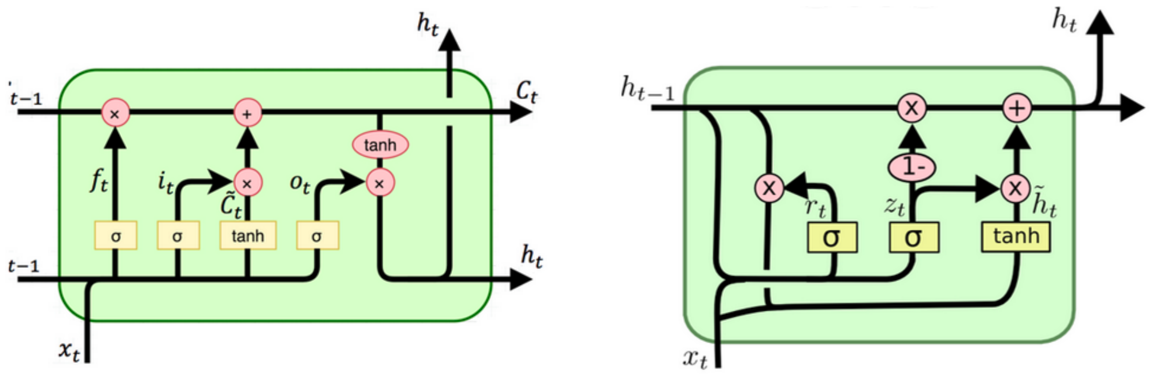
Figure 2.4: Illustration of the Vanishing Gradient problem

In order to fix the Vanishing Gradient problem, researchers have developed new, stronger methods which we will explain in the next paragraphs.

2.4.2 Long Short-Term Memory

The Long Short-Term Memory (LSTM), was introduced by Jürgen Schmidhuber *et al.* in 1997 [23] as a solution to the vanishing gradient problem. It’s a modified version of the Vanilla RNN, that can capture long-term dependencies. Its internal mechanisms that are called *gates* that can control the flow of information. These gates can learn which data in the sequence is important to keep or throw away. By doing that, it learns to use relevant information to make predictions.

The main component of the LSTM architecture is the *cell state*. The cell state is the “memory” of the network. It connects the the different layers of the model by allowing the flow of important information through it. The ability to add/remove any information is achieved with the usage of 3 gates. The first is the **Input gate**, which decides how much information from the input unit is added to the hidden state. The second is the **Output gate**, which decides how much information from the current cell makes it to the output. The third is the **Forget gate**, which decides the amount of the past information that the model shall keep in memory. Fig. 2.5a shows the structure of an LSTM cell.



(a) LSTM cell (b) GRU cell

Figure 2.5: A visualization of (a) the LSTM cell and (b) the GRU cell

The outputs at timestep t of: the input gate i_t, the output gate o_t, the forget gate f_t, pseudo cell state $\tilde{C}_t$, cell state C_t and hidden state h_t, can be computed as:

$$\begin{aligned} i_t &= \text{sigmoid}\left(x_t W_{xi} + a_{t-1} W_{hi}\right) \\ o_t &= \text{sigmoid}\left(x_t W_{xo} + a_{t-1} W_{ho}\right) \\ f_t &= \text{sigmoid}\left(x_t W_{xf} + a_{t-1} W_{hf}\right) \\ \tilde{C}_t &= \tanh\left(x_t W_{xg} + a_{t-1} W_{hg}\right) \\ C_t &= \text{sigmoid}\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right) \\ a_t &= \tanh\left(C_t\right) * o_t \end{aligned}$$

where $W_{xi}, W_{hi}, W_{xo}, W_{ho}, W_{xf}, W_{hf}, W_{xg}, W_{hg}$ are the weights of the model.

2.4.3 Gated Recurrent Unit

The Gated Recurrent Unit (GRU), was introduced in 2014 by Kyunghyun Cho *et al.* [19] as an alternative to the LSTM model. Even though at that time the LSTM was SOTA, it had a major drawback; it was really slow during training. This was a result of the number of gates. The GRU aimed to deal with this issue by reducing the number of gates from 3 to 2. The 2 gates are now named **Update gate** and **Reset gate**, and are denoted by z_t and r_t (at timestep t) respectively. Fig. 2.5b shows the structure of a GRU cell. The equations for the GRU model are:

$$\begin{aligned} z_t &= \text{sigmoid}\left(W_z * [h_{t-1}, x_t]\right) \\ r_t &= \text{sigmoid}\left(W_r * [h_{t-1}, x_t]\right) \\ \tilde{h}_t &= \tanh\left(W * [r_t * h_{t-1}, x_t]\right) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \end{aligned}$$

where W_z, W_r, W are the weights of the model.

2.4.4 Bi-directionality of RNNs

LSTMs and GRUs are useful because they can store information from the beginning of a sequence and transmit it successfully until the end. But what happens in the opposite scenario, where the end of a sequence contains information that could be used by the output of the first timesteps? The current models as we described them, form directed graphs that propagate information only in one way. This idea can be generalized in order to gain information in the opposite direction. This can be achieved by using a second model that scans and processes the sequence from end to beginning. The outputs of both models can be concatenated together to form a single bi-directional representation of any layer. A bi-directional LSTM (Bi-LSTM) is illustrated in fig. 2.6.

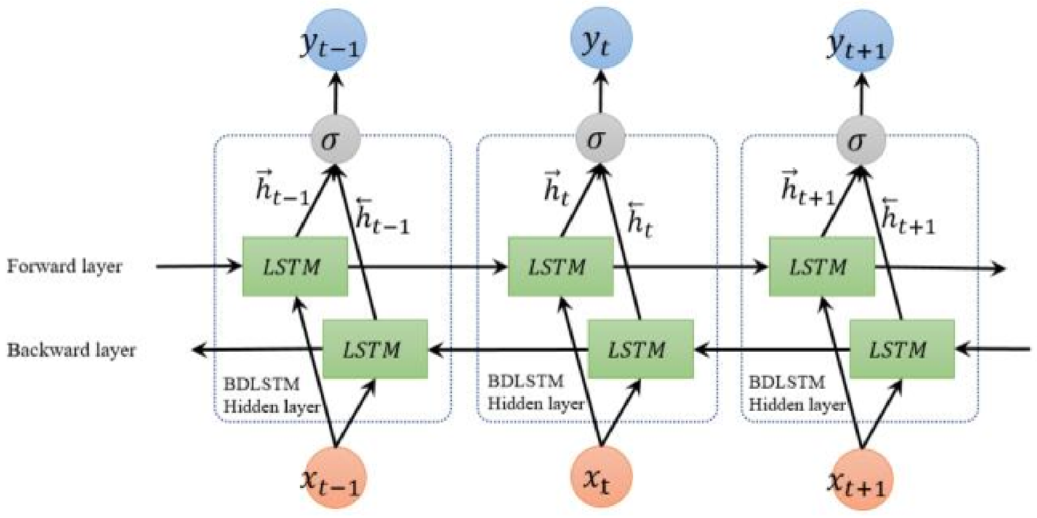


Figure 2.6: Bi-directional LSTM architecture

2.5 Encoder Decoder Architecture

The Encoder-Decoder architecture is a NN framework based on the Sequence to Sequence (Seq2Seq) modelling technique. It's basically the 4th type of RNN that was discussed earlier in figure 2.3d, where an input sequence x of fixed-length T_x gets mapped to an output sequence $\hat{y}$ of fixed-length T_y. Here, the Encoder refers to the first part of the RNN that computes the hidden states and the Decoder refers to the second part that takes these hidden states and starts generating the output sequence $\hat{y}$. This model can be generalized so that the Encoder and Decoder are separate NNs, as shown in fig. 2.7.

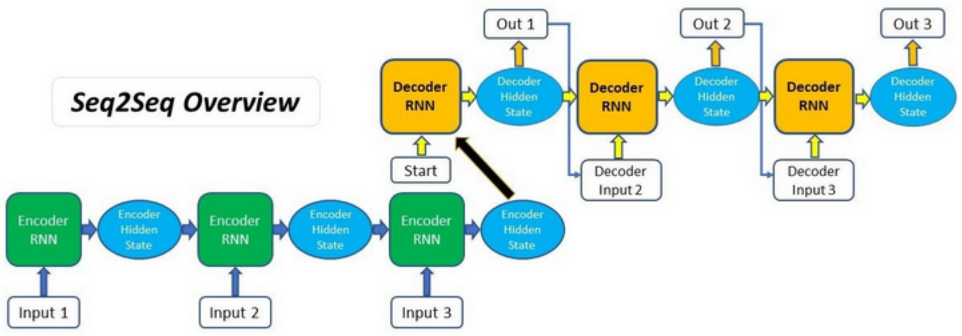


Figure 2.7: Encoder-Decoder architecture with separate RNNs

2.6 The Attention Mechanism

Even though LSTMs, GRUs and Seq2Seq models were performing extremely well for a variety of tasks, they were still far away from achieving, let alone surpassing, human-like performance in most tasks. They had some serious drawbacks that needed to be fixed in order for harder NLP problems to be solved.

2.6.1 The need for a better model

The previous approaches had 2 major disadvantages. First, they **couldn't be parallelized**. In order to compute the hidden state h_t at timestep t, the previous hidden state h_{t-1} had to be computed. The time complexity of a forward pass scaled linearly with the length of the sequence. Since many NLP tasks dealt with longer sequences (i.e. longer than 500 tokens), this was a major slowdown during training, especially when training with huge text corpora. Second, **there wasn't a way to make any token attend explicitly to any other token**. For example, consider the input sentence "John and Nick were playing table tennis against Maria and Sophia respectively." and the question "Who were the opponent pairs in the table tennis matches?". Ideally, the model should be able to match John with Maria and Nick with Sophia. A name in one pair should be able to *attend* to the other name in the pair. Since we have 2 pairs, we will have 2 different signals. But how will those 2 different signals be encoded in 1 hidden state? This might be possible for only 2 different signals, but how is this going to work for larger sequences with more than 300 tokens, where there are many different relations that need to be encoded?

2.6.2 Attention: An elegant solution

The idea to overcome these disadvantages is very simple and elegant; Match every token in a sequence with every other token. This mechanism is called *Attention*, because practically every token *attends* to every other token. This allows the model to retain and utilize all the hidden states of the of the input sequence during the decoding process. It creates a unique mapping between each timestep of the decoder output to all the encoder hidden states, providing the decoder with access to the entire sequence, and the capability to pick out specific elements from that sequence in a discriminating way, for every output it makes towards the final product.

Initially, attention was used in combination with RNNs (specifically, LSTMs) in order to tackle the problem of attending all the tokens of a sequence with all the other tokens. The hidden states would all be combined with a parameterized weighted sum into the *context vector*, which is what would then be fed in the decoder section of the model. The parameters for the weighted sum are the parameters (weights) of the attention mechanism. This is illustrated in fig. 2.8.

The aforementioned architecture managed to improve drastically the quality of RNNs, but training was still extremely slow. A new architecture had to be invented, in order to overcome the high complexity of the forward pass of RNNs, which would also allow for parallelization. Even though progress seemed to have stagnated for a bit, researchers were not done yet.

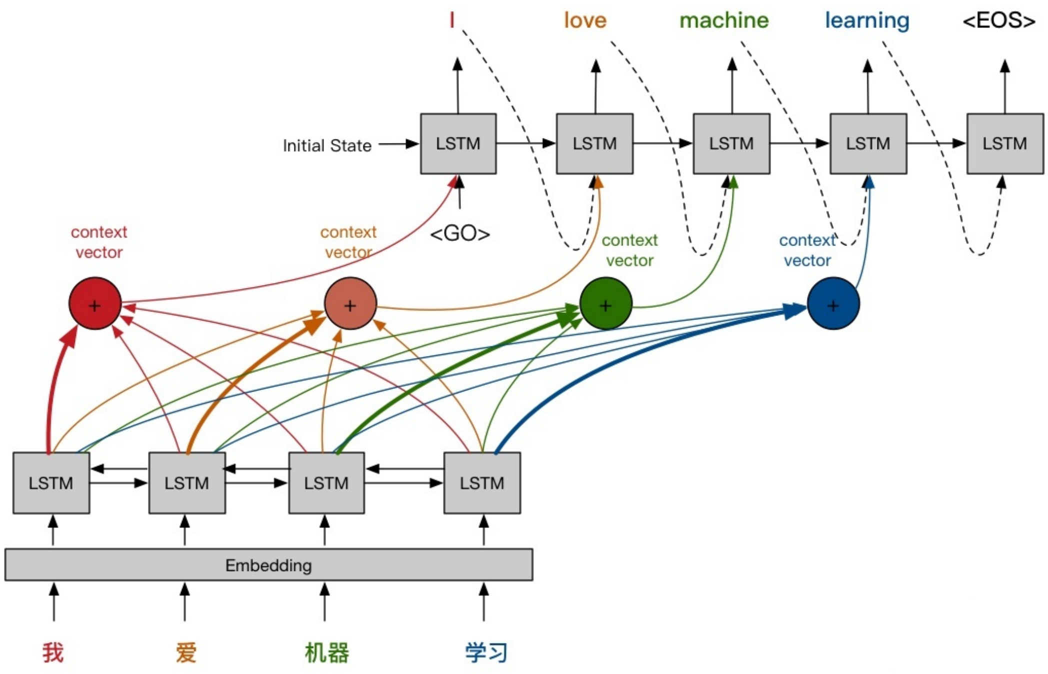


Figure 2.8: Attention Mechanism + Seq2Seq LSTM for Machine Translation

2.7 Transformers

Attention is All you Need is a paper published by A. Vaswani *et al.* in 2017 [38] that revolutionized the field of NLP. It's arguably one of the most impactful papers that have been published, as it presents the *Transformer* model, a model which shatters all previous baselines, and sets out to become the default architectural choice for LMs. As its name suggests, it gets rid of the RNNs and instead it uses only the attention mechanism.

The Transformer model is an alternative to RNNs, that is designed to handle sequential data. Although it borrows the encoder-decoder framework from seq2seq models, it differs from RNNs because it allows for data to be processed in parallel. It also introduces a new variant of the attention mechanism, called *self-attention*.

The self-attention module works by comparing every token in the sequence to every other token in the sequence, including itself, and re-weighing the embeddings of each token to include contextual relevance. It takes in n embeddings without context and returns n embeddings with contextual information. For example, in the phrase, "Bank of the river", Bank would be compared with Bank, of, the, and river, and as Bank is compared with those four words, its word embedding would be re-weighted to include the relevance of the words to its own meaning in the sentence accordingly. This module can be generalized to the *Multi-head Attention*, where many self-attention blocks run in parallel. Fig. 2.9 illustrates this mechanism.

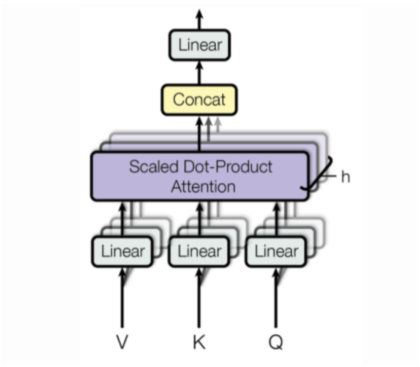


Figure 2.9: Multi-head self-attention mechanism

The Scaled Dot-Product Attention is computed as

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{d_k}\right)V \tag{2.1}$$

where Q stands for Query, K stands for Key and V stands for Value. Those vectors are the representations (embeddings) of the tokens.

The Multi-head self-attention module is the backbone of the Transformer architecture. The Transformer is made up of stacks of Encoder and Decoder blocks. The Encoder block consists of a Multi-head self-attention module followed by an MLP with a skip connection and layer normalization [15]. The Decoder block is the same, but it also consists of a Masked Multi-head self-attention (for prediction) module before the Multi-head self-attention. An illustration of this architecture can be found in fig. 2.10.

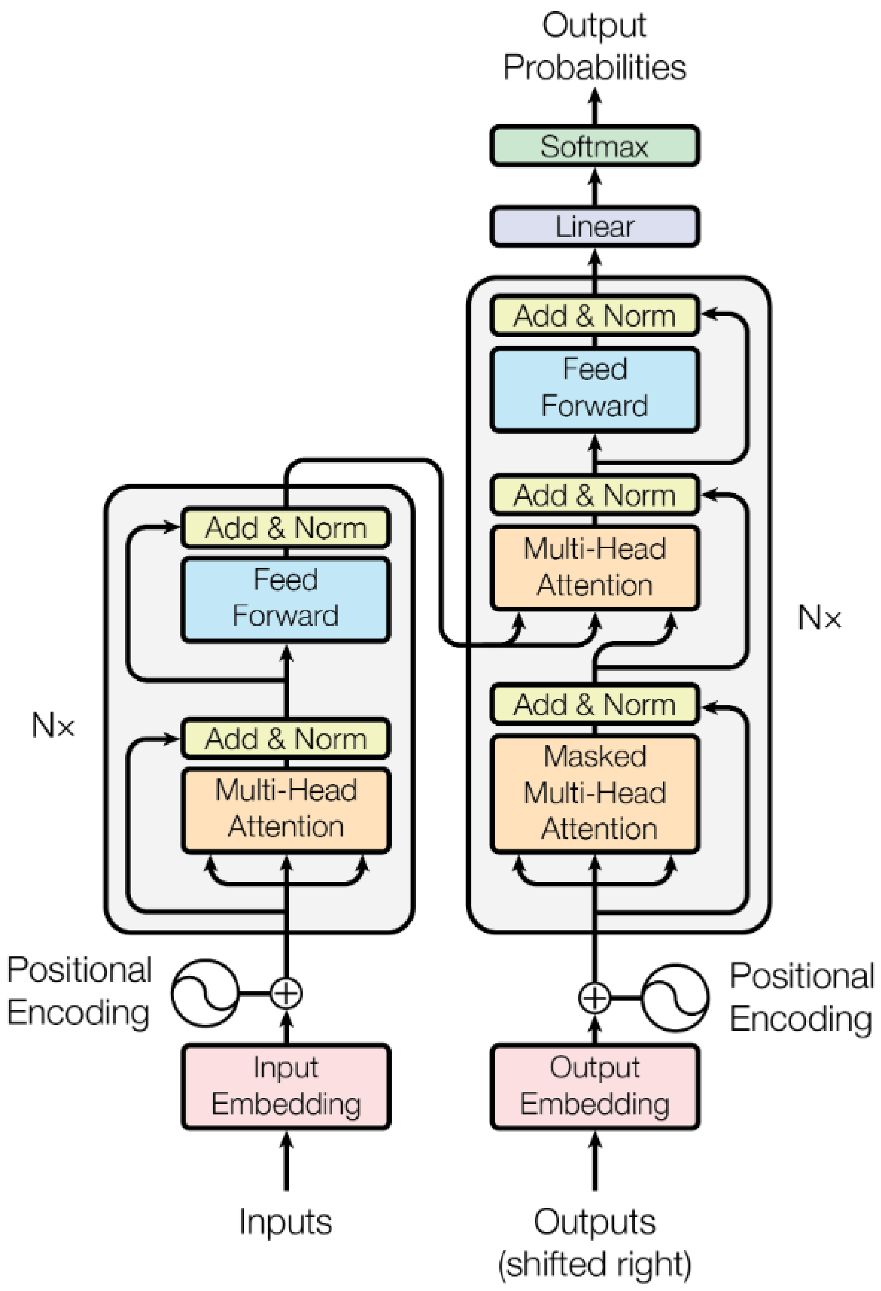


Figure 2.10: Transformers Architecture: (left) Encoder and (right) Decoder

2.8 BERT

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a Language Model that was introduced by the Google AI Language team in 2018 [21]. At the time of its release, it outperformed all previous LMs on various tasks. It also showed the importance of pre-training when fine-tuning models on downstream tasks.

The architecture of the BERT model makes use of the Transformer NN. Specifically, as its name suggests, BERT consists of a stack of many Encoders of the Transformer Architecture. The Encoders are used in order to learn useful language representations (embeddings) for each token in the input sequence. Since the Encoder architecture leverages the self-attention mechanism, context is taken into account. This means that the model might, and actually *will*, produce different representations for the same token, it it has different context, i.e. it's in a different sentence.

The $\text{BERT}_{\text{BASE}}$ model has 12 encoder layers stacked on top of each other, which sums up to 110M parameters, whereas $\text{BERT}_{\text{LARGE}}$ has 24 layers of encoders stacked on top of each other, which sums up to 340M parameters. Needless to say, these models are huge, and this is one of the reason for why they perform so well.

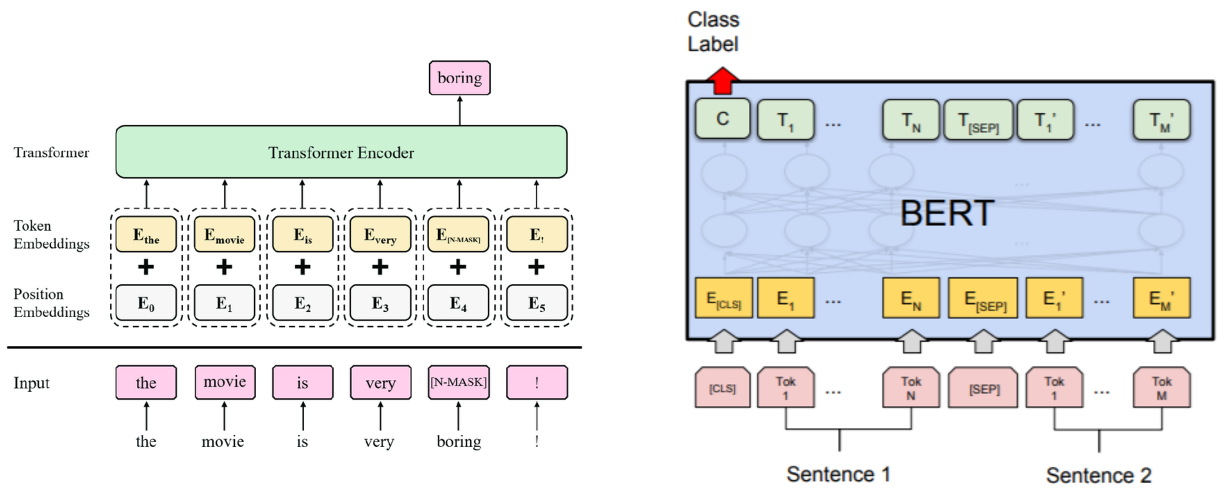
One thing that is quite important though and that played a big role in BERT achieving great results, is the procedure with which it was trained. Before BERT, most LMs were trained on the auxiliary task of Next Token Prediction. That is, they were given an incomplete sequence as input and they had to guess the next word in the sequence. For example, if the input sequence was “The quick brown fox jumps over the lazy”, then the label would be “dog”, and so the model could train in a supervised fashion to correctly predict the label. In BERT, the authors used 2 novel auxiliary tasks, called **Masked Language Modelling** and **Next Sentence Prediction**.

2.8.1 Masked Language Modelling

Masked Language Modelling (MLM) refers to the auxiliary task when tokens from a sequence are masked out, and the model is train to predict which are those tokens. This helps the model understand context, as it needs to take into account the other non-masked tokens in order to understand which token is missing. The output of the last encoder of BERT is fed to a small MLP classifier that has an output dimension equal to the vocabulary size. The model assigns a probability to each token in the vocabulary, and it's trained to minimize the cross entropy loss for the masked-out tokens. Fig. 2.11a shows this method.

2.8.2 Next Sentence Prediction

Next Sentence Prediction (NSP) refers to the auxiliary task of classifying whether 2 sequences are logically entailed. BERT uses some special tokens in its vocabulary in order to distinguish certain parts of text. It uses [CLS] in the beginning of the sequence, [SEP] at the end of a sequence, [PAD] for padding tokens, [UNK] for unknown tokens and [MASK] for masked-out tokens. In NSP, 2 sentences are fed into the model, and it has to classify whether they are entailed. This helps the model learn long-term dependencies and also capture the meaning of whole sequences. Fig. 2.11b shows this method.



(a) Masked Language Modelling (b) Next Sentence Prediction

Figure 2.11: A visualization of the auxiliary tasks and methods used to pre-train the BERT model: (a) MLM and (b) NSP

2.8.3 Leveraging the BERT Language Model

After pre-training, BERT can successfully represent almost any sentence as a concatenation of embeddings, one for every token of the input sentence. This latent representation can then be used in order to perform some downstream task, for example Question Answering. That is, a new MLP head can be added on top of the last Encoder of BERT that will learn to answer a given question, based on the embeddings produced by the BERT model. The LM will remain frozen (its parameters won't be updated) during this fine-tuning phase. This method has shown great results in the BERT paper [21].

2.9 BERT Variants

The BERT model was revolutionary. But research didn't stop there. There was still a lot of room left for improvement. First, BERT was extremely slow to train due to the huge number of parameters it had. Second, even though it achieved amazing results, they could still improve. That's why many variants of BERT that aim to deal with these drawbacks have been released. Some of the most notable ones will be introduced in the next subsections.

2.9.1 RoBERTa

RoBERTa (Robustly Optimized BERT Pretraining Approach) was introduced by Y. Liu *et al.* in 2019 [27] as an alternative to BERT. In this paper, the authors chose to stick with the previous model architecture (stacked encoders), and proposed three new ideas for the pre-training phase.

First, they **get rid of Next Sentence Prediction**. Even though the authors of BERT argue that the NSP task improves performance, further experiments in combination with the second idea of RoBERTa show that it actually decreases performance in downstream tasks. Therefore, they only keep the MLM task for pre-training.

Second, they propose **dynamic masking** for MLM. In the original BERT paper, *static masking* was used, where words in specific sentences would be masked and stay that way throughout the whole training process. *Dynamic masking* refers to masking the MLM inputs on the fly, when sampling batches. This showed an increase in performance for two downstream tasks.

Third, they **experiment with many different ideas on how to model the MLM input**. In the original BERT paper, the input was sampled as a pair of segments from the same document, where each segment could contain multiple sentences. The authors of RoBERTa tried three new methods of modelling the input. First, they tried to sample just two sentences instead of two segments. They keep NSP for this method. Second, they tried to sample as many sentences from a document that could fit in a single batch. If a sentence doesn't fit, they pad to maximum length. If the document finishes before the maximum length is achieved, they would add a separator token and start sampling sentences from the next document. Third, like in the second approach, they sample as many sentences as they could fit in a single batch, but this time they pad the rest of the batch with pad-tokens instead of sampling sentences from a new document, when a document finishes. These approaches are also known as *SENTENCE-PAIR + NSP*, *FULL-SENTENCES*, and *DOC-SENTENCES*. The latter approach seems to perform best on most downstream tasks, but the authors chose to use the second approach because it's easier to implement, more data goes through the model with every forward pass and the decrease in performance (compared to the last approach) is marginal.

2.9.2 DistilBERT

DistilBERT (a Distilled version of BERT) was introduced by V. Sanh *et al.* [34] as a lighter version of BERT. The idea is to create a model that achieves almost the same performance as BERT$_{base}$, but is much smaller and faster.

As the name suggests, the authors use **Knowledge Distillation** [18, 22], a compression technique in which a compact model -the student- is trained to reproduce the behaviour of a larger model -the teacher- or an ensemble of models. In supervised learning, a classification model is generally trained to predict an instance class by maximizing the estimated probability of gold labels. A standard training objective thus involves minimizing the cross-entropy between the model's predicted distribution and the one-hot empirical distribution of training labels [34]. Having this in mind, we can define the training objective of DistilBERT with 3 individual losses. First, BERT is distilled. That is, the outputs of BERT$_{base}$ are used as golden labels for DistilBERT. Second, it's trained in the MLM objective. And third, a cosine embedding loss is used because it helps in aligning the directions of the student and teacher hidden states vectors.

The results are very promising, as DistilBERT, which has half the number of encoders that BERT$_{base}$ has, shows that it is possible to reduce the size of BERT$_{base}$ by 40%, while retaining 97% of its language understanding capabilities and being 60% faster.

2.9.3 ALBERT

ALBERT (A Lite BERT) was introduced by Z. Lan *et al.* [26] as an alternative to the pre-training steps of BERT. It uses novel methods for pre-training LMs and it surpasses the previous benchmarks set by BERT.

The contributions that the paper proposes are mainly 3-fold: Factorized embedding parameterization, cross-layer parameter sharing, and intern-sentence coherence loss. The first two address the issue of model size and memory consumption in BERT; the third corresponds to a new auxiliary task in pre-train, Sentence Order Prediction, replacing the NSP task in BERT.

The first contribution, **Factorized embedding parameterization**, refers to the technique of decomposing the WordPiece embeddings into two smaller matrices. Quoting the paper: "Instead of projecting the one-hot vectors directly into the hidden space of size H (hidden layer size), they are projected into a lower dimensional embedding space of size E (WordPiece Embedding size), and then projected it to the hidden space. By using this decomposition, the embedding parameters are reduced from $\mathcal{O}(V \times H)$ to $\mathcal{O}(V \times E + E \times H)$, where V is the vocabulary size. This parameter reduction is significant when $H \gg E$, which is the case in ALBERT."

The second contribution, **Cross-layer parameter sharing**, refers to sharing all all parameters of the network across all layers. Even though there are many ways to share parameters inside a NN (e.g., only sharing feed-forward network (FFN) parameters across layers or only sharing attention parameters), the parameter efficiency improves when parameters are shared in all layers.

The third contribution, **Sentence Order Prediction** (SOP), refers to a new auxiliary task in pre-training, where the model receives as input two consecutive segments, either normally (positive examples) or swapped (negative examples), and has to classify which type they are. That is, the difference with NSP is that in negative examples, NSP samples two sentences randomly, which is way easy for the model to distinguish. SOP forces the model to learn finer-grained distinctions about discourse-level coherence properties, thus improving the overall performance of the model.

ALBERT, like other BERT variants, introduces novel ideas and manages to improve performance, while reducing drastically the model size. It has a few drawbacks, with one of the most important ones being speed. The more "powerful" models, ALBERT$_{\text{xlarge}}$ and ALBERT$_{\text{xxlarge}}$, record a 0.6$\times$ and 0.3$\times$ slowdown compared to BERT$_{\text{LARGE}}$.

3. DEVELOPMENT OF ANCIENT GREEK LANGUAGE MODEL

In this chapter, we will discuss the process of finding and gathering plain-text data, the training setup, the hyperparameters of the model and some other implementation specific details.

3.1 Data

There are three main steps regarding data that need to be taken in order to start developing a LM: Gathering, pre-processing and training a tokenizer.

3.1.1 Gathering the data

Finding data in the form of Ancient Greek plain-text was definitely one of the most challenging parts of this journey. Simply because there isn't enough publicly available data. Most BERT-like LMs have been trained on corpora containing at least 15-20GB of text. The publicly available data which we were able to leverage is roughly 440MB, and it came from Perseus, First1kGreek Project, Portal and text from the Diorisis Ancient Greek Corpus [7, 3, 9, 1]. The relevant statistics can be seen in table 3.1.

Table 3.1: Statistics on pre-training corpora for the Ancient Greek LM

Corpus	Size (MB)	Number of sentences
Perseus	114.2	410K
First1kGreek	230	928K
Portal	3.5	9.9K
Diorisis	89.7	323K
Total	437.4	1672K

Thesaurus Linguae Graecae (TLG) [10] is a website that contains texts from over 2750 authors. Sadly, it is mentioned in their website that the data can't be downloaded. When they were asked to provide us with access to their database for research purposes, they kindly refused. Thus, we had to work with data downloaded from the aforementioned sources. In fig. 3.1a we can see a sample of how the data from these sources looks like.

```
Πολλάκις ἐθαύμασα τίσι ποτὲ λόγοις Ἀθηναίους ἔπεισαν οἱ γραψάμενοι Σωκράτην ὡς ἄξιος εἴη
θανάτου τῇ πόλει. ἡ μὲν γὰρ γραφὴ κατ’ αὐτοῦ τοιάδε τις ἦν· ἀδικεῖ Σωκράτης οὓς μὲν ἡ πόλις
νομίζει θεοὺς οὐ νομίζων, ἕτερα δὲ καινὰ δαιμόνια εἰσφέρων· ἀδικεῖ δὲ καὶ τοὺς νέους διαφθείρων.
```

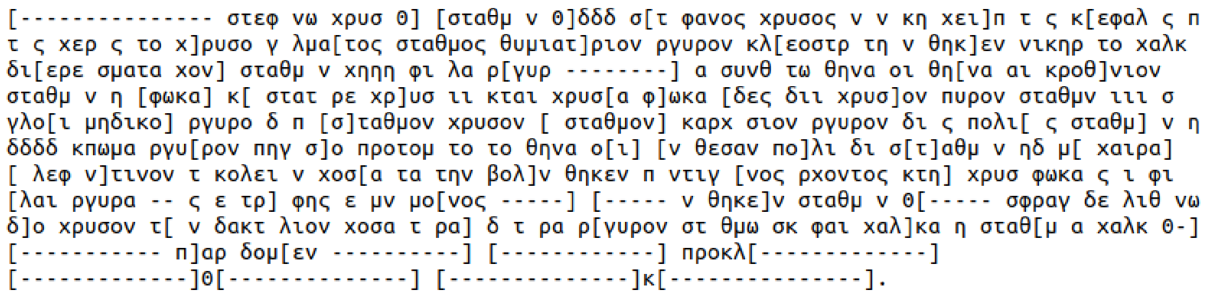
(a) A sample of Ancient Greek text

```
πολλακις εθαυμασα τισι ποτε λογοις αθηναιους επεισαν οι γραψαμενοι σωκρατην ως αξιος ειη
θανατου τη πολει. η μεν γαρ γραφη κατ’ αυτου τοιαδε τις ην· αδικει σωκρατης ους μεν η πολις
νομιζει θεους ου νομιζων, ετερα δε καινα δαιμονια εισφερων· αδικει δε και τους νεους διαφθειρων.
```

(b) The same sample, but pre-processed

Figure 3.1: Raw Ancient Greek text (a) and its pre-processed version (b).

Furthermore, there is data of Ancient Greek inscriptions made available by the Packard Humanities Institute (PHI) [8]. This data was used in Deepminds model called *Ithaca* [14], which was trained to restore missing text from inscriptions. Unfortunately, as can be seen in fig. 3.2, the data is not in a form that could be useful to train a LM, due to the stray letters that is has and the amount of text that is missing. Therefore, we chose not to include it in our training corpora.



```
[--------------- στεφ νω χρυσ θ] [σταθμ ν θ]δδδ σ[τ φανος χρυσος ν ν κη χει]π τ ς κ[εφαλ ς π
τ ς χερ ς το χ]ρυσο γ λμα[τος σταθμος θυμιατ]ριον ργυρον κλ[εοστρ τη ν θηκ]εν νικηρ το χαλκ
δι[ερε σματα χον] σταθμ ν χηηη φι λα ρ[γυρ --------] α συνθ τω θηνα οι θη[να αι κροθ]νιον
σταθμ ν η [φωκα] κ[ στατ ρε χρ]υσ ιι κται χρυσ[α φ]ωκα [δες διι χρυσ]ον πυρον σταθμν ιιι σ
γλο[ι μηδικο] ργυρο δ π [σ]ταθμον χρυσον [ σταθμον] καρχ σιον ργυρον δι ς πολι[ ς σταθμ] ν η
δδδδ κπωμα ργυ[ρον πηγ σ]ο προτομ το το θηνα ο[ι] [ν θεσαν πο]λι δι σ[τ]αθμ ν ηδ μ[ χαιρα]
[ λεφ ν]τινον τ κολει ν χοσ[α τα την βολ]ν θηκεν π ντιγ [νος ρχοντος κτη] χρυσ φωκα ς ι φι
[λαι ργυρα -- ς ε τρ] φης ε μν μο[νος -----] [----- ν θηκε]ν σταθμ ν θ[----- σφραγ δε λιθ νω
δ]ο χρυσον τ[ ν δακτ λιον χοσα τ ρα] δ τ ρα ρ[γυρον στ θμω σκ φαι χαλ]κα η σταθ[μ α χαλκ θ-]
[----------- π]αρ δομ[εν ----------] [------------] προκλ[-------------]
[-------------]θ[--------------] [--------------]κ[---------------].
```

Figure 3.2: A sample of data from PHI inscriptions

Apart from the MLM data, there is also the Part-of-Speech (POS) Tagging data. This was taken from the Diorisis dataset. There is a lot of noise in this data, that's why a big fraction of the dataset has been filtered out. Still, some outliers remain, but they are so few and therefore it shouldn't affect training. The available POS tags are *adjective*, *adverb*, *article*, *conjunction*, *interjection*, *noun*, *numeral*, *particle*, *preposition*, *pronoun*, *proper*, *verb* and *punct*. Table 3.2 shows the amount of labels per class (PoS tag).

Table 3.2: Statistics on the Diorsis Corpora POS data

PoS Tag	Train	Validation	Test	Total
adjective	989K	12K	3K	100K
adverb	451K	6K	2K	459K
article	961K	10K	3K	976K
conjunction	684K	9K	2K	695K
interjection	19K	499	224	20K
noun	1021K	12K	3K	1038K
numeral	767	0	0	767
particle	667K	8K	2K	678K
preposition	477K	5K	2K	484K
pronoun	622K	8K	3K	634K
proper	273K	2K	1K	277K
verb	1514K	20K	6K	1542K
punct	1037K	13K	5K	1056K

3.1.2 Pre-processing the data

Before talking about pre-processing, it is worth mentioning that the raw downloaded data needs to be cleaned, as it has quite some noise (stray hyphens, stanza indices, names of people participating in dialogues, etc..). This process is carried automatically after downloading the data.

Ancient Greek characters have many variations due to the existence of accents. The accents do not contribute to the semantic meaning of the corresponding word or sentence, as they are mainly used in speech. Therefore, they can be removed. Also, the case of the letters does not affect the semantics. Therefore, all letters can be converted to lowercase. A sample python code for the implementation of these pre-processing steps can be found in the snippet below.

```python
import unicodedata

def clean_text(text: str) -> str:
    """Cleans the given text by stripping accents and lowercasing."""
    non_accent_characters = [
        char for char in unicodedata.normalize('NFD', text)
        if unicodedata.category(char) != 'Mn'
    ]
    # str.lower() works for unicode characters
    return ''.join(non_accent_characters).lower()
```

The effect of this method can be seen in fig. 3.1b, which is a pre-processed version of 3.1a.

3.1.3 Training a tokenizer

This step is pretty straightforward thanks for the Huggingface community [4] that has already build a library for this task, called *tokenizers* [11]. Specifically, the tokenizer "Byte-LevelBPETokenizer" was used, which makes use of Byte-Pair Encoding (BPE) [36], a technique while relies on a pre-tokenizer that splits the training data into words. In table 3.3, the hyperparameters (along with what they represent) of the tokenization process can be found.

Table 3.3: Hyperparameters of the tokenization process

Hyperparameter	Explanation
vocab_size	The maximum size of the vocabulary.
min_freq	The minimum frequency a token must have in order to be added to the vocabulary.

3.2 Model

All the BERT-like architectures that were discussed in the previous chapter, have more or less the same potential. Their difference in performance is noticeable, yet marginal. This means that if one model "works", then so should the other models, and vice-versa. Due to this reason, and after some experimentation, we decided to use RoBERTa, with the *DOC-SENTENCES* formatting of the input. A detailed list of hyperparameters for the pre-training and the architecture of RoBERTa can be found in table 3.4.

Table 3.4: Hyperparameters for the RoBERTa LM

Hyperparameter	Explanation
max_length	The maximum length of a single sequence.
mask_prob	The probability that each token of a sequence is masked in the mini-batch (dynamic masking).
num_hidden_layers	Number of hidden layers in the Transformer encoder.
num_attention_heads	Number of attention heads for each attention layer in the Transformer encoder.
hidden_size	Dimensionality of the encoder layers and the pooler layer.
intermediate_size	Dimensionality of the “intermediate” (feed-forward) layer in the Transformer encoder.
hidden_act	The non-linear activation function in the encoder and pooler.
hidden_dropout_prob	The dropout probability for all fully connected layers in the embeddings, encoder, and pooler.
attention_probs_dropout_prob	The dropout ratio for the attention probabilities.
max_position_embeddings	The maximum sequence length that this model might ever be used with.
layer_norm_eps	The epsilon used by the layer normalization layers.
classifier_dropout	The dropout ratio for the classification head.

It’s also important to take a look at the hyperparameters of the training process, as they are more crucial for the performance of the model. They can be seen in table 3.5.

Table 3.5: Hyperparameters for the training process of the RoBERTa LM

Hyperparameter	Explanation
batch_size	Mini-batch size during training.
learning_rate	The learning rate of the Adam optimizer.
weight_decay	L2 regularization strength.
scheduler_factor	The factor by which to reduce the learning rate when the training loss starts plateauing.
scheduler_patience	Number of steps (updated) with no progress to wait in order to apply the reduction in the learning rate.
train_epochs	Number of epochs (passes through all the data) to train the model for.

3.2.1 Setup

The python programming language was used. The models were developed with the PyTorch Deep Learning framework. The models were trained on 4 Nvidia Tesla V100 GPUs. The training time per epoch was roughly 20 minutes for larger models.

4. EXPERIMENTS AND RESULTS

There are way too many hyperparameters in the end-to-end process. In order to reduce the search space, we decided to fix some that don't affect performance that much, and some where good values have been found empirically. The default values for all the hyperparameters are provided in table 4.1. Wherever a specific value of a hyperparameter is not mentioned in an experiment, then the default value from this table is being used.

Table 4.1: Default values for the hyperparameters.

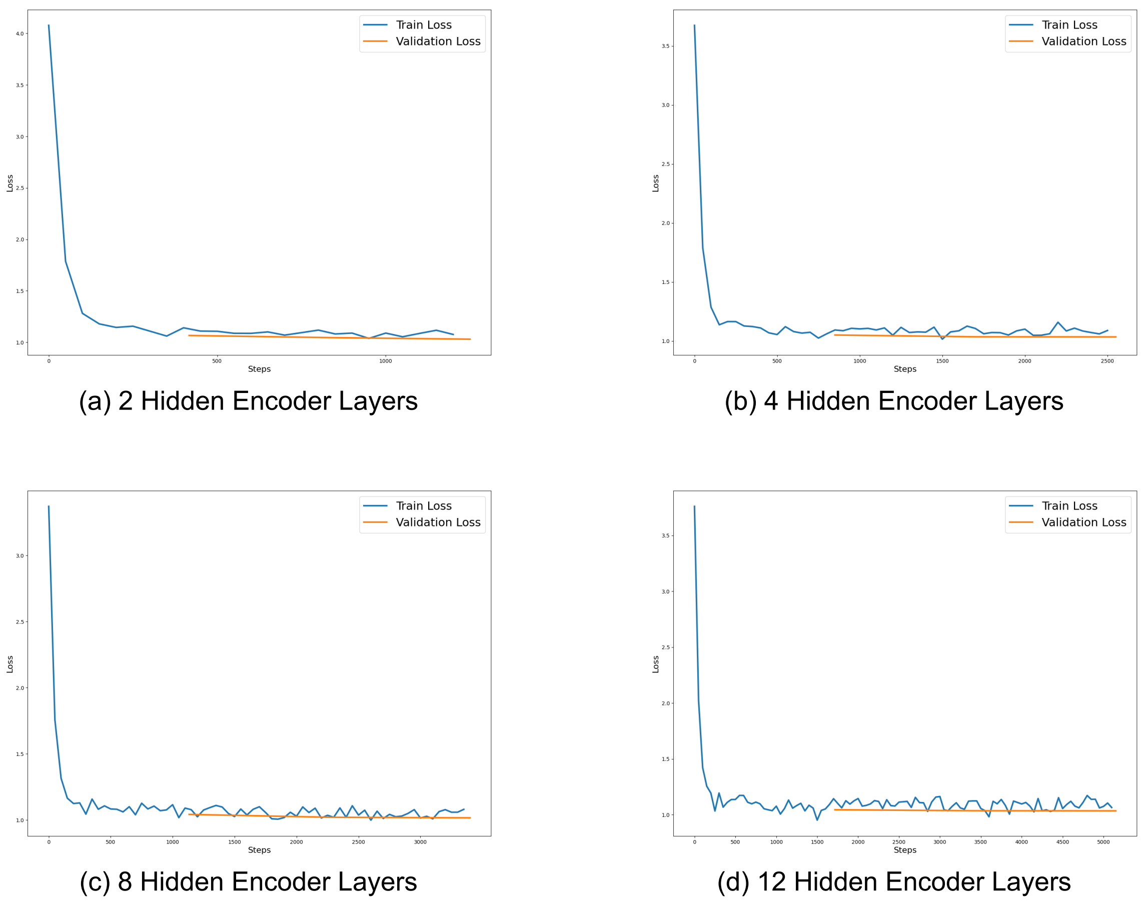
Hyperparameter	Default value
vocab_size	30522
min_freq	2
max_length	512
mask_prob	0.15
num_hidden_layers	6
num_attention_heads	12
hidden_size	768
intermediate_size	3072
hidden_act	GELU
hidden_dropout_prob	0.1
attention_probs_dropout_prob	0.1
max_position_embeddings	512
layer_norm_eps	10^{-12}
classifier_dropout	0
batch_size	32
learning_rate	10^{-4}
weight_decay	0.01
scheduler_factor	0.1
scheduler_patience	10
train_epochs	3

4.1 Impact of Hidden encoder layers

Since the amount of data is extremely limited, it could be possible that a smaller architecture is needed for the model to not under-perform. Various experiments were held with different number of hidden encoder layers. The results for some of them can be seen in table 4.2 and fig. 4.1.

Table 4.2: Results for different number of Encoder Layers.

Loss	2 Encoders	4 Encoders	8 Encoders	12 Encoders
Train	1.05	1.06	**1.04**	1.05
Validation	1.02	1.03	**1.01**	1.03



(a) 2 Hidden Encoder Layers

(b) 4 Hidden Encoder Layers

(c) 8 Hidden Encoder Layers

(d) 12 Hidden Encoder Layers

Figure 4.1: Learning curves or different values of the number of hidden encoder layers.

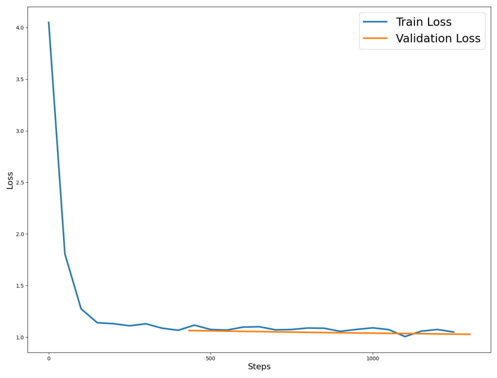
As we can see from both the table and the figures, the number of Hidden encoder layers does affect the performance of the model. The batch size was adjusted to the maximum number of samples that could fit into memory. Still, no model manages to achieve a good cross-entropy loss.

4.2 Impact of Attention heads

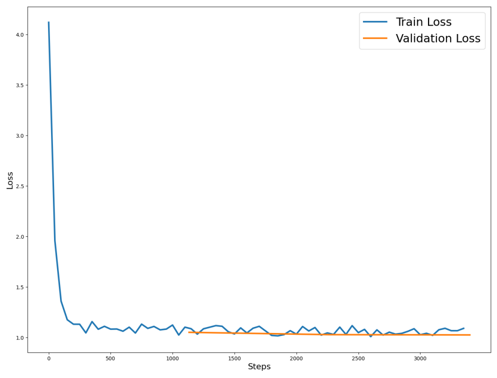
The number of attention heads (in the Multi-head self attention mechanism) also greatly defines the the model size. Most models use the default value which is 12, which has been found empirically to work well. Yet, we still make more experiments to see if better performance is achieved by lowering this number. Table 4.3 and fig. 4.2 show the results for those experiments.

Table 4.3: Results for different number of Attention Heads and Encoder Layers.

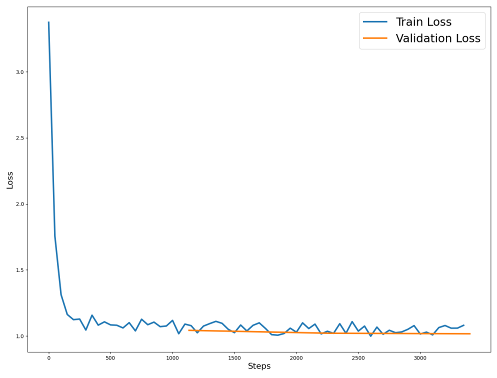
Configuration	Train Loss	Validation Loss
8 Attention Heads, 2 Encoders	1.05	1.02
16 Attention Heads, 2 Encoders	1.06	1.03
8 Attention Heads, 8 Encoders	**1.04**	**1.01**
16 Attention Heads, 8 Encoders	1.04	1.01



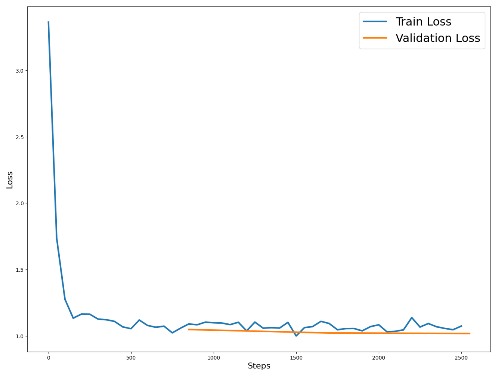
(a) 8 Attention Heads, 2 Encoders



(b) 16 Attention Heads, 2 Encoders



(c) 8 Attention Heads, 8 Encoders



(d) 16 Attention Heads, 8 Encoders

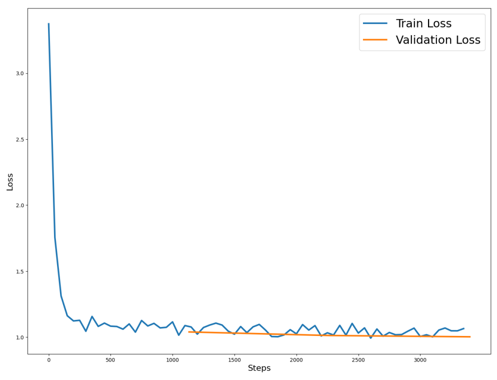
Figure 4.2: Learning curves for different values of the number of Attention heads and the number hidden encoder layers.

4.3 Impact of Learning rate

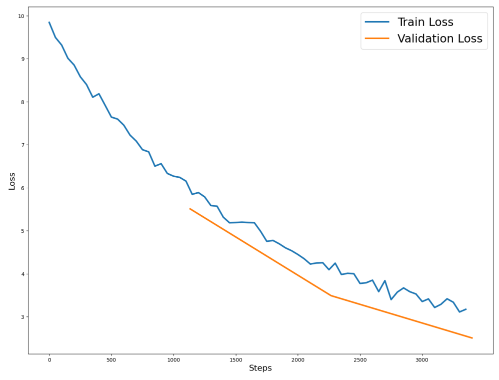
Arguably one of the most important hyperparameters during training is the learning rate. Most BERT-based model have used the Adam Optimizer with a learning rate of 10^{-6}. In some approaches, it was initially warmed up to 10^{-4} and then decayed to 10^{-6}. Also, L2 regularization was used, with a weight decay of 0.01, again for most models. We will stick with the weight decay of 0.01 in our experiments, and try to change the learning rate and its scheduling. Table 4.4 and fig. 4.3 show the results. Note that these results are for the model which performed best in the previous experiments.

Table 4.4: Results for different values of Learning rate and scheduling, for the model with 8 Hidden Encoder Layers and 8 Attention heads.

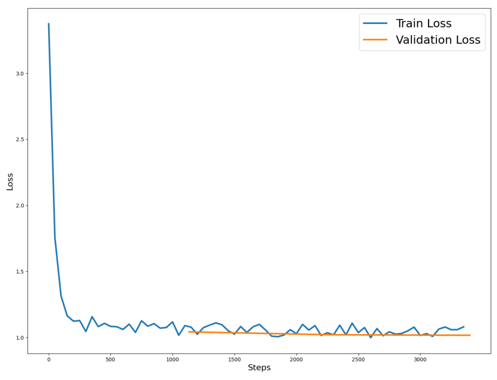
Configuration	Train Loss	Validation Loss
10^{-4} Learning rate, no scheduling	**1.04**	**1.00**
10^{-6} Learning rate, no scheduling	3.11	2.55
10^{-4} Learning rate, with scheduling	1.04	1.01
10^{-6} Learning rate, with scheduling	3.17	2.50



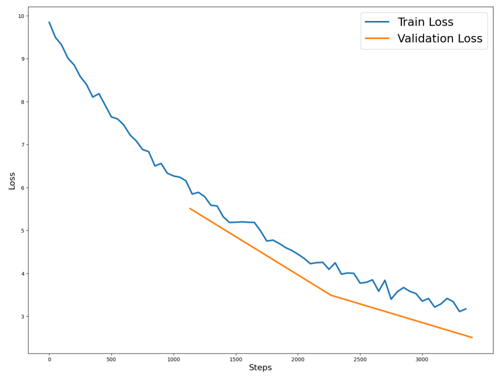
(a) 10^{-4} Learning rate, no scheduling



(b) 10^{-6} Learning rate, no scheduling



(c) 10^{-4} Learning rate, with scheduling



(d) 10^{-6} Learning rate, with scheduling

Figure 4.3: Learning curves for different values of the learning rate and its scheduling.

4.4 Bayesian Optimization

Bayesian optimization was also tried in order to find better hyperparameter values. Specifically, the Hyperopt [5] python library was used. A total of 50 evaluations were performed using the Tree of Parzen Estimators (TPE), where the objective function to be minimized was the validation loss. Again, no significant improvements were found from the search, so the results are omitted. The search space can be seen in table 4.5.

Table 4.5: Bayesian Optimization Search Space.

Hyperparamater	Type	Search space
num_hidden_layers	quniform	2 - 12
num_attention_heads	quniform	6 - 12
hidden_size	choice	[256, 512, 768, 1024]
batch_size	choice	[4, 8, 16, 32]
learning_rate	loguniform	$\log\left(10^{-7}\right)$ - $\log\left(10^{-2}\right)$
weight_decay	loguniform	$\log\left(10^{-3}\right)$ - 0

4.5 Error Analysis

The best validation loss that was achieved is that of 1.00. For Cross-Entropy, this still remains a high value. It would be interesting though to take a look at what the model has learnt. Tests showed that the models are skewed towards punctuation, as it appears way more often than other tokens. Also, it is skewed towards common tokens such as “και” and “το”. Different regularization techniques were tried in order to overcome this, such as increasing the dropout rate in the encoders and the L2 weight decay, but none improved significantly the results.

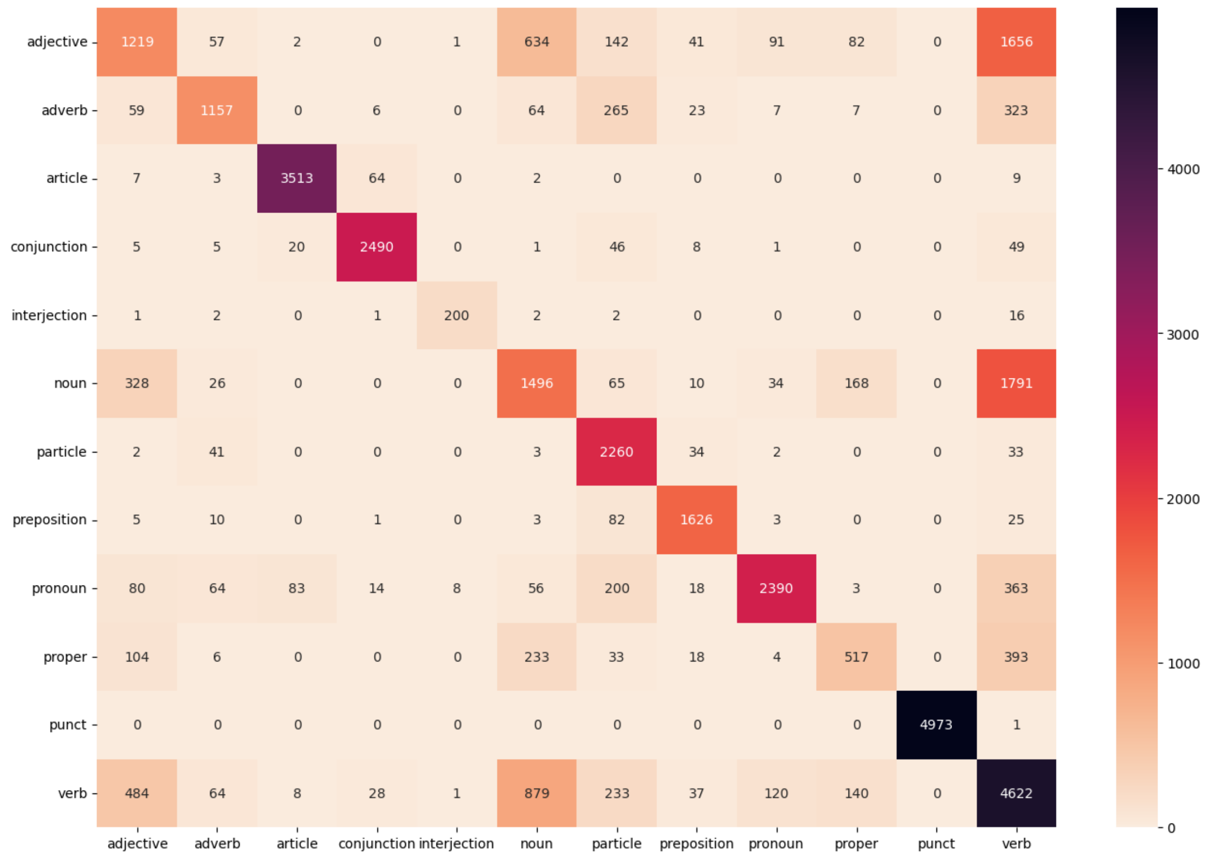
4.6 PoS Tagging

At this point, since no LM managed to achieve very good performance during pre-training, it doesn’t make much sense to fine-tune a model for a downstream task. Yet, for the sake of completeness, we used the pre-trained model with the lowest validation loss, added a MLP head on top of the last encoder and fine-tuned it for PoS Tagging on the Diorisis dataset. Table 4.6 shows the recall and f1 score for each class on the test set.

Table 4.6: Unseen test set results for the PoS Tagging fine-tuned model.

PoS Tag	Recall	F1-score
adjective	0.31	0.39
adverb	0.61	0.69
article	0.98	0.97
conjunction	0.95	0.95
interjection	0.89	0.92
noun	0.38	0.41
numeral	-	-
particle	0.95	0.79
preposition	0.93	0.91
pronoun	0.73	0.80
proper	0.40	0.47
verb	0.99	0.99
punct	0.70	0.58

Even though the Diorisis dataset is a bit noisy, the results are pretty decent. The model is capable of identifying most verbs, articles, conjuctions, interjections and prepositions. Of course, this task is not that difficult since in languages such as Ancient Greek, the suffix of a word usually is enough to define it’s Part-of-Speech Tag. Yes, this is a solid way to better evaluate the performance of a LM. Fig. 4.4 shows the confusion matrix for the test set and fig. 4.5 shows the result for a sentence randomly sampled from the test set.



	adjective	adverb	article	conjunction	interjection	noun	particle	preposition	pronoun	proper	punct	verb
adjective	1219	57	2	0	1	634	142	41	91	82	0	1656
adverb	59	1157	0	6	0	64	265	23	7	7	0	323
article	7	3	3513	64	0	2	0	0	0	0	0	9
conjunction	5	5	20	2490	0	1	46	8	1	0	0	49
interjection	1	2	0	1	200	2	2	0	0	0	0	16
noun	328	26	0	0	0	1496	65	10	34	168	0	1791
particle	2	41	0	0	0	3	2260	34	2	0	0	33
preposition	5	10	0	1	0	3	82	1626	3	0	0	25
pronoun	80	64	83	14	8	56	200	18	2390	3	0	363
proper	104	6	0	0	0	233	33	18	4	517	0	393
punct	0	0	0	0	0	0	0	0	0	0	4973	1
verb	484	64	8	28	1	879	233	37	120	140	0	4622

Figure 4.4: Confusion Matrix for the Diorisis PoS test set.

```
In [18]: def print_results(tag_items, labels, str_, le):
             print(f'{"Token":10}\t\t\t{"Truth":10}\t\t\t{"Predicted":10}')
             print('------------------------------------------------------------------------------')
             for tag_item, label in zip(tag_items, labels):
                 if label != -100:
                     y_hat = int(tag_item['entity'].split('_')[1])
                     pos_tag = le.inverse_transform([y_hat])[0]
                     label = le.inverse_transform([label])[0]
                 else:
                     pos_tag = '-'
                     label = '-'
                 token = str_[tag_item['start']:tag_item['end']]
                 print(f'{token:10}\t\t\t{label:10}\t\t\t{pos_tag:10}')
```

```
In [19]: tagger = TokenClassificationPipeline(model=pos_model, tokenizer=tokenizer)
         pos_tags = tagger(' ο δεδο τεκαι παθοιλω και την μη τραπεζαν γδ κωνσταντιου γενε.')
         labels = [2, 7, 9, -100, 12, -100, 3, 2, -100, 0, -100, 12, 9, 11]
         print_results(pos_tags, labels, text, label_encoder)
```

```
Token                           Truth                           Predicted
------------------------------------------------------------------------------
ο                               article                         article
δε                              particle                        particle
δο                              pronoun                         pronoun
τεκαι                           -                               -
παθοι                           verb                            verb
λω                              -                               -
και                             conjunction                     conjunction
την                             article                         article
μη                              -                               -
τραπεζαν                        adjective                       verb
γδ                              -                               -
κωνσταντιου                     verb                            verb
γενε                            pronoun                         pronoun
.                               punct                           adjective
```

Figure 4.5: PoS Tags for randomly sampled sentence from the test set. When creating constructing the sentences from individual tokens, some may be split in many sub-tokens during the process of tokenization. In this case, the label "-" is used for the previous sub-tokens and the actual PoS tag is used for the last sub-token.

4.7 Discussion

Surprisingly, the results for PoS Tagging are reasonably good. The LM are clearly underfitting, due to the small amount of data they are seeing. The Ancient Greek language changed a lot over the course of history. This means that there is a lot of diversity in the language. In order for a model to capture the essence and understand in depth the language, it has to be trained with a much larger corpora. Also, to make sure that those results were not caused by some bug in the code, the Trainer API from Huggingface was used, which produced similar results.

Yet, by looking at the training curves, we can see that the model is definitely learning something. This is also backed up by the fact that the model performs decently when fine-tuned for the downstream task of Part-of-Speech Tagging. This encourages us to believe that a combination of a larger corpora and a better architecture and/or pre-training methods would definitely produce better results.

5. CONCLUSION AND FUTURE WORK

In this thesis, we attempt to develop a BERT-like Language Model for the Ancient Greek Language, and then fine-tune it for the downstream task of Part-of-Speech tagging. Unfortunately, the results confirm exactly what the theory suggests: Huge Language Models need huge corpora to achieve good results. In our case, we had a bit less than 450 MB of plain text data at our availability, which is a few orders of magnitude below what other Language Models have used for their corresponding languages.

An extensive hyperparameter search was performed, in order to make sure that that was not the issue. We tried hand-tuning many values, and in the end we also tried using bayesian optimization. The performance didn't improve much. We also tried different models sizes, but again it help that much.

Yet, despite the huge lack of data, we managed to fine-tune a model for the downstream task of Part-of-Speech tagging, and the results were reasonably good. Having this in mind and by looking at the learning curves, we can safely say that the model is learning something. The validation loss decreases a lot but it converges to a value that is not good enough. This is exactly what theory says: The model is underfitting due to lack of data. This means that if more data will be gathered in the future, the same model could potentially perform much better. That's why the code for this thesis has been made available at `https://github.com/AndrewSpano/BSc-Thesis`.

One idea that is of particular interest, is to see if the current version of the (modern) Greek-BERT could be edited by some editing method such as MEND [29] in order to be fine-tuned for some downstream task in Ancient Greek. Greek-BERT has been trained on a much larger corpora and has shown good performance, which makes this idea very appealing.

ABBREVIATIONS - ACRONYMS

AI	Artificial Intelligence
ML	Machine Learning
NLP	Natural Language Processing
SOTA	State-of-the-art
TL	Transfer Learning
ELMo	Embeddings from Language Models
BERT	Bidirectional Encoder Representations from Transformers
GPT	Generative Pre-trained Transformer
LM	Language Model
ANN	Artificial Neural Network
NN	Neural Network
MLP	Multi Layer Perceptron
FFNN	Feed Forward Neural Network
UFA	Universal Function Approximator
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
Bi-LSTM	Bi-directional LSTM
GRU	Gated Recurrent Unit
Seq2Seq	Sequence to Sequence
MLM	Masked Language Modelling
NSP	Next Sentence Prediction
SOP	Sentence Order Prediction
BPE	Byte-Pair Encoding
POS	Part of Speech
NER	Named Entity Recognition
NLL	Negative Log Likelihood
TLG	Thesaurus Linguae Graecae
PHI	Packard Humanities Institute
TPE	Tree of Parzen Estimators

BIBLIOGRAPHY

[1] Diorisis ancient greek corpus. https://figshare.com/articles/dataset/The_Diorisis_Ancient_Greek_Corpus_JSON_/12251468. Accessed: 2022-03-16.

[2] Exponential growth of AI research in 2020. https://www.zeta-alpha.com/post/growth-of-ai-research-in-2020-steady-on-the-exponential-path-in-times-of-crisis. Accessed: 2022-03-16.

[3] First1kgreek. https://github.com/OpenGreekAndLatin/First1KGreek. Accessed: 2022-03-16.

[4] Huggingface. https://huggingface.co/. Accessed: 2022-03-16.

[5] hyperopt. https://github.com/hyperopt/hyperopt. Accessed: 2022-03-16.

[6] Part-of-speech tagging. https://www.sketchengine.eu/blog/pos-tags/. Accessed: 2022-03-16.

[7] Perseus. https://github.com/PerseusDL/canonical-greekLit. Accessed: 2022-03-16.

[8] Phi. https://inscriptions.packhum.org/allregions. Accessed: 2022-03-16.

[9] Portal. https://www.greek-language.gr/greekLang/ancient_greek/tools/corpora/translation/contents.html. Accessed: 2022-03-16.

[10] Tlg. hhttp://stephanus.tlg.uci.edu/. Accessed: 2022-03-16.

[11] Tokenizers repository. https://github.com/huggingface/tokenizers. Accessed: 2022-03-16.

[12] Ancient-greek-char-bert. https://github.com/brennannicholson/ancient-greek-char-bert, 2020. Accessed: 2022-03-16.

[13] Armen Aghajanyan, Akshat Shrivastava, Anchit Gupta, Naman Goyal, Luke Zettlemoyer, and Sonal Gupta. Better fine-tuning by reducing representational collapse. *arXiv preprint arXiv:2008.03156*, 2020.

[14] Yannis Assael*, Thea Sommerschield*, Brendan Shillingford, Mahyar Bordbar, John Pavlopoulos, Marita Chatzipanagiotou, Ion Androutsopoulos, Jonathan Prag, and Nando de Freitas. Restoring and attributing ancient texts using deep neural networks. *Nature*, 2022.

[15] Jimmy Ba, Jamie Kiros, and Geoffrey Hinton. Layer normalization. 07 2016.

[16] David Bamman and Patrick J. Burns. Latin BERT: A contextual language model for classical philology. *CoRR*, abs/2009.10053, 2020.

[17] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020.

[18] Cristian Buciluundefined, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’06, page 535–541, New York, NY, USA, 2006. Association for Computing Machinery.

[19] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[20] Gobinda G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51–89, January 2003.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[22] Geoffrey Hinton, Jeff Dean, and Oriol Vinyals. Distilling the knowledge in a neural network. pages 1–9, 03 2014.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[24] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[25] John Koutsikakis, Ilias Chalkidis, Prodromos Malakasiotis, and Ion Androutsopoulos. Greek-bert: The greeks visiting sesame street. *11th Hellenic Conference on Artificial Intelligence*, Sep 2020.

[26] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019.

[27] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[28] A. H. Miller, W. Feng, A. Fisch, J. Lu, D. Batra, A. Bordes, D. Parikh, and J. Weston. Parlai: A dialog research software platform. *arXiv preprint arXiv:1705.06476*, 2017.

[29] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. In *International Conference on Learning Representations*, 2022.

[30] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[31] XiPeng Qiu, TianXiang Sun, YiGe Xu, YunFan Shao, Ning Dai, and XuanJing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897, Sep 2020.

[32] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[33] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[34] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[35] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In Stefanos D. Kollias, Andreas Stafylopatis, Włodzisław Duch, and Erkki Oja, editors, *Artificial Neural Networks – ICANN 2006*, pages 632–640, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[36] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909, 2015.

[37] Pranaydeep Singh, Gorik Rutten, and Els Lefever. A pilot study for bert language modelling and morphological analysis for ancient and medieval greek. In *The 5th Joint SIGHUM Workshop on Computational Linguistics for Cultural Heritage, Social Sciences, Humanities and Literature (LaTeCH-CLfL 2021)*, 2021.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[39] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.