Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Øyvind Steensland

Fine-Tuning BERT for Document Ranking

Master's thesis in Applied Physics and Mathematics
Supervisor: Thiago Guerrera Martins
June 2021

NTNU
Norwegian University of
Science and Technology

Øyvind Steensland

Fine-Tuning BERT for Document Ranking

Master's thesis in Applied Physics and Mathematics
Supervisor: Thiago Guerrera Martins
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

Document ranking is the task of ranking a list of documents based on a user query such that the most relevant documents come first. This is used in many applications, for example search engines or scientific databases. In this thesis, we investigate how BERT can be used to solve this problem. BERT is a machine learning model published by Google in 2018, based on the Transformer architecture. It has been pre-trained to understand natural language and has quickly become the state of the art within language understanding.

In order to use BERT for document ranking, we fine-tuned it using the MS MARCO document ranking dataset. The chosen model was a binary classifier that tries to predict whether a document is relevant to the query or not, thus creating a relevance score for the document. To take documents as input, each document was split into overlapping passages and the average passage score was used as the document score.

After testing the model on 200 queries from MS MARCO, our results show that BERT significantly outperforms the BM25 baseline, improving performance by over 10%. The performance correlates with the number of parameters and bigger models are able to improve performance further. Our experiments also show that increasing the number of candidate documents does not increase ranking performance.

Based on the findings in the thesis, we conclude that BERT is able to use its language understanding to find relevance between query and document, making it appealing for information retrieval systems. To deal with BERT's slow speed, the use of knowledge distillation techniques is able to improve performance, while reducing the inference times.

Sammendrag

Dokumentrangering handler om å rangere en liste dokumenter basert på en søketekst slik at de mest relevante dokumentene kommer øverst på lista. Dette brukes blant annet i søkemotorer eller vitenskapelige databaser. I denne oppgaven utforsker vi om BERT kan brukes til å løse dette problemet. BERT er en maskinlæringsmodell som ble publisert av Google i 2018 og er basert på transformere. BERT har blitt forhåndstrent til å forstå naturlig språk og har raskt blitt ledende inne språkforståelse.

For å bruke BERT for dokumentrangering, finjusterte vi modellen på MS MARCO, Microsofts eget dokumentrangeringsdatasett. Den valgte modellen var en binær klassifiseringsmodell som prøver å forutsi hvorvidt et dokument er relevant for søketeksten eller ikke, og dermed lager en poengsum for hvert dokument. For at modellen skal kunne ta inn dokumenter, ble hvert dokument delt opp i overlappende passasjer og gjennomsnittsummen av alle passasjene ble satt som dokumentets poengsum.

Etter å ha testet modellen på 200 søketekster fra MS MARCO, viste resultatene våre at BERT utkonkurrerer BM25 og forbedrer ytelsen med over 10%. Ytelsen korrelerer med antall parametere og større modeller kan øke ytelsen ytterliggere. Eksperimentene våre viste også at det å øke antall kandidatdokumenter ikke øker ytelsen.

Basert på funnene i denne masteroppgaven, konkluderer vi med at BERT er i stand til å bruke sin språkforståelse til å finne relevans mellom søketekst og dokument, noe som gjør den attraktiv for informasjonsgjenfinningssystemer. For å ta hånd om BERTs lave fart er kunnskapsdestillasjon i stand til å både øke farten og forbedre ytelsen.

Preface

This thesis marks the completion of my master's degree in Industrial Mathematics at the Norwegian University of Science and Technology (NTNU). The project was done at the Department of Mathematical Sciences and supervised by Thiago Guerrera Martins.

I would like to thank my supervisor, Thiago, for suggesting the thesis subject and providing guidance throughout the semester. All the help with the Vespa software was really appreciated and I am grateful for you answering all my questions. I had no experience with IR or NLP beforehand, so I have learned a lot from working on this project and found the subject very interesting and challenging.

I would also like to thank the HPC group at NTNU for granting access to the Idun cluster (Själander et al. 2019). The extra computation power was really useful and made it much easier to train models. I would recommend it to other students at NTNU who plan to do a project that needs some extra computing resources.

Øyvind Steensland
Trondheim, June 10, 2021

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Text search applications are something most people rely on for their day-to-day activities. Every time we search for something on Google, try to find a restaurant on Tripadvisor, or find a Christmas gift on eBay, we are relying on fast and accurate search engines. Within these applications, there is a complex machinery that works to give us back results that satisfy our information need in the best possible way. But how does the system know which documents, web pages or articles that are relevant for us?

To find answers to that question, artificial intelligence (AI) is a prominent place to look. Natural language processing (NLP), the subfield within AI that deals with how computers understand natural language, has made great advancements in the last decade with the thriving success of deep learning (Eisenstein 2018, p. 47). Advancements in computation power combined with increased amounts of data have facilitated the transition from classical methods to more advanced machine learning models.

In October 2018, a newcomer with the name *BERT* entered the stage. BERT is a Google-developed language model that has been pre-trained to understand human language. Based on theory and methods from transfer learning, BERT can be fine-tuned to solve a wide range of language-related tasks. It was created out of the *Transformer architecture*, which is based solely on attention mechanisms and therefore introduces a new way computers understand language (Devlin et al. 2019).

In this thesis, we explore how BERT can be used as a part of a text search application and more specifically, *document ranking*. In the document ranking problem, we want to retrieve a ranked list of documents based on the input of a user. The goal is to sort the documents such that the documents that come first are the ones that meet the user's need the most.

1.1 Motivation

The main motivation to use BERT is based on the impressive success it has achieved on numerous language understanding problems, outperforming many state-of-the-art models. BERT outperforms *all* other models on all the problems in the General Language Understanding Evaluation benchmark (GLUE) (Devlin et al. 2019). GLUE is a collection of nine problems that is used to measure how models like BERT are able to understand language. Given BERT's proven language understanding, the goal is that it can be used to rank documents by actually understanding how the user's need relates to a document.

Based on this principle, BERT-based models were quickly adopted to ranking applications. The results were yet again remarkable and BERT-based ranking models achieved significant improvements compared to classical methods (Lin et al. 2020, p. 67). However, there is still a lot of uncharted territory to explore, which motivates further research on the topic.

The document ranking problem is a challenge within the field of information retrieval (IR), which serves as the foundation behind all text search applications. By showing that BERT works to rank documents, it suggests the use of BERT within other IR tasks, such as question answering (QA) and information filtering for example. As a result, research on the topic is also beneficial for purposes outside of the document ranking scope.

The use of BERT within document ranking is not limited to academic purposes. A year after its publication, Google announced in a blog post how they are using BERT in their search engines to improve user experience. With BERT, the search engine becomes better at grasping subtle nuances in the search texts, which is one of the major drawbacks of classical methods that match keywords.

1.2 Research Questions

The main objective in the thesis is to explore how BERT can be used for document ranking. To break down the objective, we present the following research questions that we aim to answer in the thesis. To find answers to the questions, we create our own text search application and fine-tune our own BERT models before conducting several experiments. All the details about the methodology are found in chapter 4.

Research Question 1 – *How well do BERT-based document ranking models perform compared to classical, well-used ranking methods?*

This question is the most basic and creates the foundation for the subsequent questions. When answering this question, we want to compare how our results line up with the current findings on the topic. This lets us know that our setup is working and helps to substantiate the results from the other experiments.

Research Question 2 – *What are important factors when fine-tuning BERT for document ranking?*

An important part of the BERT methodology is how the model is fine-tuned to a specific task. We therefore want to investigate how some factors from the fine-tuning impact the performance.

Research Question 3 – *How do factors that influence the speed of BERT impact the ranking performance?*

The biggest drawback with BERT is its slow inference speed. Knowing how we best can balance speed and performance is therefore crucial for many ranking systems with limited computational resources, as well as improving the user experience by returning search results quickly.

1.3 Structure of Thesis

The thesis contains the following chapters:

- **Chapter 2 – Background Theory:**
  This chapter presents all the background theory we need to understand the research topic, the methods and models that were used in addition to anything else that is needed to know to understand the rest of the thesis. The main focus is the BERT model and theory from information retrieval.

- **Chapter 3 – Related Work:**
  Here, recent work that has been done on the topic is described. This includes different models and ideas that have been used to rank documents with BERT, as well as the current ranking performances. The content of this chapter motivates many of the choices made in the subsequent chapters and lets us compare our results to the findings of similar research.

- **Chapter 4 – Method:**
  A thorough presentation of the research methodology is given, including the way data was collected, how the models were evaluated and some rationale for the choices that were made.

- **Chapter 5 – Experiments and Results:**
  In this chapter, all experiments which aim to answer the research questions are presented and the associated results are shown. After each experiment, the results are discussed.

- **Chapter 6 – Conclusion and Future Work:**
  This chapter aims to wrap up the thesis, summarizing the main findings and how they relate to the research questions and related research. Lastly, some thoughts and ideas for future work are presented.

Chapter 2

Background Theory

The two main topics in the thesis are, as the title suggests, BERT and document ranking. To fully understand them, we provide the essential background theory which becomes important for the subsequent chapters. To understand the background theory, it is assumed that the reader is familiar with core concepts from statistics and linear algebra.

The first part of the chapter is concerned with theory from information retrieval. We explain how document ranking is done in practice and some classical methods are presented. Metrics to evaluate information retrieval systems are presented, as well as well-used datasets.

The rest of the chapter introduces the concepts that we need to know to understand BERT. This includes machine learning and natural language processing, where previous methods are introduced to motivate the creation of the Transformer. BERT was created based on the concepts that the Transformer introduced and hence it plays a key role in the revolution that has happened within NLP. We have therefore devoted an entire section to explain the most important concepts from the Transformer model. The background theory chapter culminates with the presentation of the BERT model.

2.1 Information Retrieval

The main problem we are trying to solve in this thesis comes from *information retrieval* (IR), which is a field within computer science. Manning et al. (2009) define information retrieval as:

> Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

In other words, IR is the foundation upon which many text search applications and search engines are built upon, and it is going to play a central role in this thesis.

2.1.1 Basic Terminology

We start by introducing some basic terminology (Manning et al. 2009):

- *Term*: A unique word or concept (for example 'New York') that appears in a piece of text. All the terms in a language make up a *dictionary*.
- *Document*: A piece of information we are retrieving. Examples are books, web pages or research papers. We denote a document as D.
- *Corpus*: The set of all documents we are searching through to retrieve information.
- *Query*: Representation of the user's information need.
- *Relevance*: A measure of how well a document satisfies the user's need.
- *Index*: A data structure containing information about the corpus that allows for increased search speed. The simplest index is the *forward index*, which is a list of all the documents in the corpus, where each document contains a list of all the terms it is made up of. Correspondingly, we have the *inverted index*, which is a list of all the terms in the dictionary, and for each term there is a list of all documents that contain the term, called a postings list. A visual representation of these indexes is shown in figure 2.1.

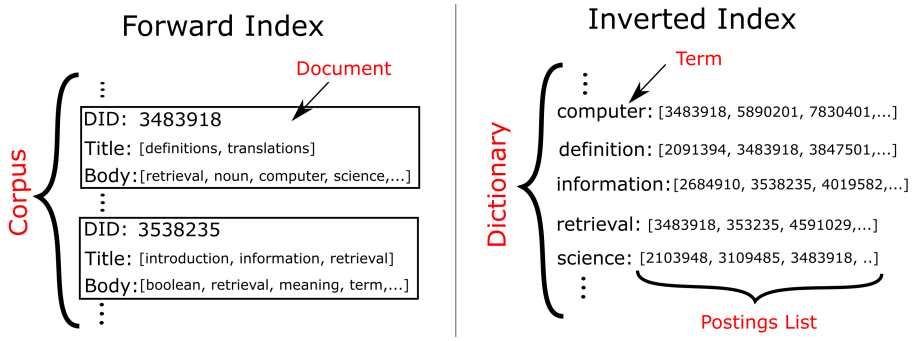


Figure 2.1: A visualization of how a corpus can be stored in a forward and inverted index.

2.1.2 Document Ranking

IR tries to solve many different problems, and in this thesis, we study the *document ranking* problem, also called ad hoc retrieval. The challenge within document ranking is to retrieve a list of documents based on the information need of the user, and then rank the list such that the most relevant documents come first. This is a key concept within search engines, and with the large increase of digital information available, a well-performing and fast document ranking system is important.

Many applications used for document ranking, especially those who have implemented BERT, use a *retrieve-and-rank* approach (Lin et al. 2020, p. 45). In this approach, we first do a search to find candidate documents, using a fast and simple method, which is called the *retrieval phase*. We then apply one or more expensive ranking models to the candidate documents, giving us a ranked list as output. We refer to this as the *ranking phase*. This is the phase we focus on in this thesis.

The main reason for splitting the search in two is simple – drastically reduce search time. Ranking documents is time consuming, so we want to restrict this phase to as few documents as possible. However, it is also important that the relevant documents are among the candidate documents, so the retrieval method cannot be too restrictive either. The number of retrieved documents and the quality of the retrieval method are therefore key to balancing ranking performance and search time.

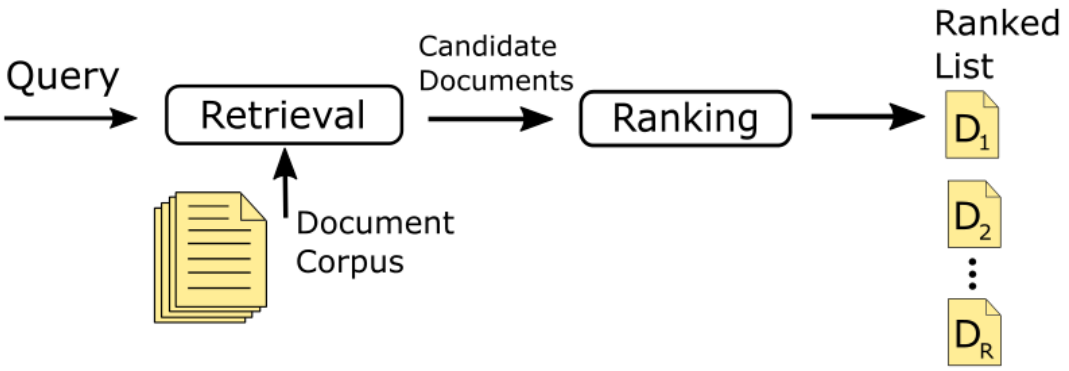


Figure 2.2: The retrieve-and-rank approach used for document ranking

2.1.3 Retrieval Methods

In this thesis, we focus on the ranking phase, but having some insight into how the retrieval is done is still relevant. We therefore present some classical retrieval methods.

Boolean Retrieval

The earliest and simplest method is probably the *boolean retrieval model*. This method makes use of boolean logic and the associated operators AND, OR and NOT. Using these, users can create boolean queries, which puts restrictions on which documents we want to retrieve. Examples of such queries are "England AND Football", retrieving all documents containing both the terms England and football or "Football OR Soccer", where documents containing either term are retrieved (Manning et al. 2009, p. 4-13).

The biggest advantage of the model is that the implementation and runtime are very fast. The method makes use of the inverted index and retrieves the postings lists of all the terms in the query. If there is an OR operator between two terms, we take the union of the two lists and the intersection in the case of an AND operator. If we see a NOT operator, we simply remove all documents containing the term.

On the other hand, the method requires a good boolean query to work well, and it does not retrieve any partial matches. We usually want the model to work on free-text queries, which are the type of queries we will have to consider for a general-purpose text search application. We therefore define the following boolean retrieval models which can be used on free-text queries:

- **OR** model: retrieve every document that contains at least one query term (i.e. an OR operator between every term)
- **AND** model: retrieve every document that contains all the query terms (i.e. an AND operator between every term)

If the query is long or if it contains common words, we would risk matching too many documents by using **OR**. This could be mitigated by the inclusion of stop words (words like "a" and "the"), but the number of matches will still be quite big. On the other hand, **AND** would, more often than not, be too restrictive and we risk not matching enough documents. For both methods, an increasing query length would further contribute to these disadvantages.

Vector Space Model

The *vector space model* was created to deal with the shortcomings of the boolean model. Firstly, it calculates a relevance score for each document, which means that we can choose how many documents we want to retrieve (we could also use this score to rank the documents, but we often want to use a more complex model for this). Secondly, it works well with free-text queries (Datta 2010).

The key idea in the model is to look at documents as vectors, where a document D_j is represented as

$$\mathbf{D}_j = \left[w_{1j}, w_{2j}, \ldots, w_{tj} \right],$$

where t is the number of terms in the dictionary. Unlike the boolean model, the vector elements are not binary but given a weight. This extracts more information about the document, giving a more nuanced picture. The vector space model is a *bag-of-words* method, meaning that the order of the terms does not matter. Note that this means two different documents could end up having the same vector representation, which could be problematic.

The most common way to calculate the term weights is to use a *td-idf* measure. It is given as the product between the *term frequence* (tf) and the *inverse document frequency* (idf). Term frequency is simply the number of times a term appears in a document. We denote this as $TF(i, D)$ for a term i and document D. Therefore, if a document contains many instances of a term, it is likely that the document has something to do with that term.

On the other hand, we do not want to weigh common words high. Words that appear in almost all documents, like "the" or "a", carry very little meaning and we want to reduce the impact they have. This is done by considering the inverse document frequency, which in its simplest form is written as:

$$IDF(i) = \log \frac{N}{n_i},$$

where N is the total number of documents and n_i is the number of documents containing the term i. As we will see later, there exist other ways to formulate the expression, but they all achieve the same objective.

Put together, the *tf-idf* measure values terms that appear often in a document, but that is not contained in many other documents. In other words, it is a simple way to extract information about what a document is really about.

In the same way as the documents, the query is also converted into a vector, denoted $\mathbf{q}$. The vector space model uses the query and document vectors to find how relevant a document is to a query by calculating their *similarity*. The standard similarity metric is the *cosine similarity*, which we write as

$$\text{sim}(\mathbf{q}, \mathbf{D}) = \frac{\mathbf{q} \cdot \mathbf{D}}{\|\mathbf{q}\|_2 \|\mathbf{D}\|_2} \in [0, 1]. \tag{2.1}$$

In other words, it is the cosine of the angle between the two vectors, where closer to 1 means more similar. After having calculated the similarity between the query and all the documents, we can retrieve the top K documents.

BM25

Similar to the vector space model is the BM25 ranking function, BM meaning "best matching". It is also a bag-of-words model and can be used both to retrieve documents and it also works well as an initial ranking model. It was created as a part of the *probabilistic relevance framework* by Stephen Robertson et. al in the 1970-1980s, which deals with trying to estimate the probability of a document being relevant (Robertson & Zaragoza 2009).

The formula for calculating the BM25 relevance score is normally written as

$$BM25(q,D) = \sum_{q_i \in D} IDF(q_i) \frac{TF(q_i,D) \cdot (k_1+1)}{TF(q_i,D) + k_1(1-b+b \cdot \frac{|D|}{avdl})}, \tag{2.2}$$

where q is the query containing the terms $q_1, \ldots, q_n$. The different components of the function are:

- The IDF is written in a little bit different form as previously, now formulated as

$$IDF(q_i) = \ln\Big(1 + \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}\Big).$$

  The reason for the change of the formula is to ensure that we do not divide by 0 or take the logarithm of 0.

- $B = 1 - b + b \cdot \frac{|D|}{avdl}$ is called soft length normalization, and it ensures that long documents do not get unreasonably high scores. Here, $|D|$ is the length of D in number of terms, while $avdl$ is the average length of all the documents. $b \in [0,1]$ is a parameter that adjusts how much the length of the document is compensated for, where $b = 0$ switches compensation off, and $b = 1$ performs full length normalization. Typical values are $b \in (0.5, 0.8)$.

- k_1 is a tuning parameter that adjusts the importance of the term frequency. The higher this value is, the more influential the term frequency is. Typical values are $k_1 \in (1.2, 2)$.

As we see, the formula is an extension of the *tf-idf* weighting measure. However, that formula would favor long documents, so we therefore try to mitigate this effect by using soft length normalization. Secondly, BM25 also allows us to control how much we want the term frequency to influence the score. These two features result in BM25 outperforming the vector space model (Datta 2010, p.22).

2.1.4 Evaluation Metrics

When making an information retrieval system, it is essential to have metrics to evaluate the models. The metrics also have to allow us to separately evaluate each part of the system. Having good metrics allows us to compare different setups, architectures and methods, which is important for the group developing a system.

For each test query, we consider a ranked list containing R elements, and average over many queries to get a complete evaluation of a model (thus adding an M in front of the metric). Additionally, we often want to evaluate the metric at a specific cutoff, $k \leq R$, and the metric obtained at that cutoff is denoted by Metric@k (Lin et al. 2020, Manning et al. 2009).

Precision and Recall

We start off with the two simplest metrics, *precision* and *recall*, which are defined by

$$\text{Precision} = \frac{\#\big(\text{retrieved relevant documents}\big)}{\#\big(\text{retrieved documents}\big)} = P\big(\text{relevant} \mid \text{retrieved}\big) \tag{2.3}$$

$$\text{Recall} = \frac{\#\big(\text{retrieved relevant documents}\big)}{\#\big(\text{relevant documents}\big)} = P\big(\text{retrieved} \mid \text{relevant}\big) \tag{2.4}$$

Together, precision and recall are simple metrics to use and easy to interpret. A good model should be able to achieve both good recall and precision, but these metrics often trade off against each other. If you want to achieve high recall, you can increase the number of retrieved documents, but this also means reducing precision. Often, whether to prioritize one over the other depends on the application of the model.

Precision and recall do not take the order of the documents into account. To evaluate how good a model is at doing ranking, we need metrics that take the order into consideration.

Average Precision

We can still make use of the precision and recall values and combine them, which is what we do when we calcuate *average precision* (AP). Average precision is calculated as

$$\text{AP} = \frac{\sum_{i=1}^{R} \text{Precision@}i \cdot \text{rel}(i)}{\#\big(\text{relevant documents}\big)}, \tag{2.5}$$

where rel(i) is an indicator function with value 1 if the document at position i is relevant, and 0 else. Note that the denominator is the total number of relevant documents, including those that are not among the top R documents.

The name average precision comes from the fact that we average the precision values at all recall levels. Hence, it captures the aspects of both precision and recall in a single metric.

Reciprocal Rank

Another ranking metric is the *reciprocal rank* (RR). We calculate it as

$$RR = \frac{1}{\text{rank}_i}, \tag{2.6}$$

where rank_i is the rank of the best ranked, relevant document. If no relevant documents are retrieved, RR is 0.

As opposed to AP, RR only takes the best ranked, relevant document into account. This makes it suitable for cases when we only have one relevant document, or when we primarily care about the best document. In the case of only one relevant document, the two coincide.

Normalized Discounted Cumulative Gain

Both AP and RR consider the case of binary relevance, where documents are considered relevant or not relevant to a query. However, in many cases, we have graded relevance, and this is where we can make use of *normalized discounted cumulative gain* (nDCG). In order to define it, we first introduce the discounted cumulative gain (DCG)

$$\text{DCG} = \sum_{i=1}^{R} \frac{\text{rel}_i}{\log_2(i+1)}, \tag{2.7}$$

where rel_i is the relevance level of the document at position i. Alternatively, in many implementations the numerator is changed to $2^{\text{rel}_i} - 1$, which puts a stronger emphasis on retrieving the most relevant documents. The two formulas coincide in the binary case.

Using DCG, we calculate the normalized DCG as

$$\text{nDCG} = \frac{\text{DCG}}{\text{IDCG}}, \tag{2.8}$$

where IDCG is the DCG we get when the list is sorted in the best possible way.

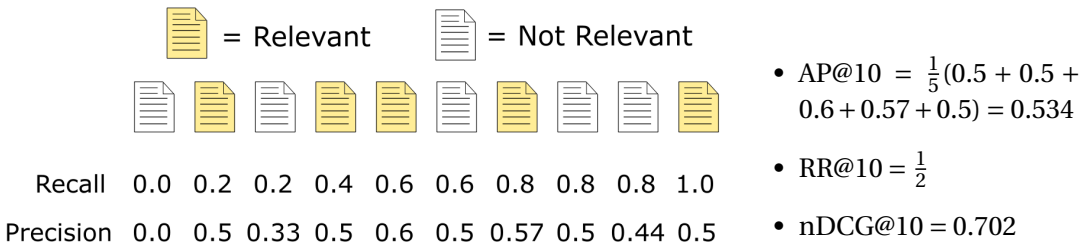


Figure 2.3: Example of how the different metrics are calculated

2.1.5 Datasets

Datasets are also an essential part of creating text search applications and models to solve the document ranking problem. They are important in the development of a model, especially to train a supervised machine learning model, but also to test the performance of a model against other models.

MS MARCO

The MicroSoft MAchine Reading COmprehension (MS MARCO) dataset collection was released in 2016, and quickly became popular. The collection contains several datasets, which can be used for related tasks such as passage ranking or question answering, but in our case, we only care about the document ranking dataset (Bajaj et al. 2018).

MS MARCO was specifically designed for deep learning purposes and is therefore of special interest to us. The huge size of the dataset makes it suitable to use to train large language models like BERT. The training dataset contains 3.2 million documents and 367013 queries, where each query has *one* relevant document, with a binary relevance judgment. It also contains 5793 test queries, where the relevant document is not given, which is used to create a leaderboard of the best models. At the moment of writing, many BERT-based models are to be seen among the top positions of the leaderboard.

The queries have been sampled from Bing search queries, and the relevant documents have been chosen by a human editor. Using this approach, the queries are as "natural" as possible, reflecting the type of queries we expect the model to be used on. It also reflects the distribution of the information need of users.

TREC Datasets

The *Text Retrieval Conference* (TREC) is a conference within IR that focuses on specific research areas, called *tracks*. In this regard, they also offer valuable datasets that researchers can use. The MS MARCO dataset collection was a part of the 2019 Deep Learning track for example.

A well-studied track is the Robust track, which focuses on creating retrieval systems that work across many different query topics. The track presents the Robust04 test collection, which is used as a benchmark to compare a lot of different document ranking models. The collection contains 249 test queries and a corpus with 528000 documents, containing high-quality relevance judgments which makes it as natural as possible (Lin et al. 2020, p. 34).

If we want to develop applications to be used on the internet, there are several web test collections available. For example, the ClueWeb09 dataset contains over a billion web pages and is used for large-scale search engines. The GOV2 test collection is also used to evaluate models, containing 25 million documents.

2.2 Machine Learning

Machine learning (ML) is a branch of artificial intelligence that uses statistical methods to find complex patterns in large amounts of data. This way, an ML model is able to generalize and make predictions about data it has not seen before. Machine learning is very applicable and is today used to solve many problems, such as object detection, speech recognition and fraud detection (Goodfellow et al. 2016, p. 96).

We can divide ML into two main branches – *supervised* and *unsupervised* learning. Supervised learning deals with models and algorithms where the data we have available is labeled, i.e. we know what the correct output for a given input is. In unsupervised learning, on the other hand, we are only given the input, and these tasks are therefore harder. Furthermore, we can divide models based on the type of data we are outputting. If we are dealing with continuous data, we call it *regression*, and in the case of categorical data (data which is separated into groups), we refer to it as *classification*. In this thesis, we will be working with classification using supervised learning.

Common for almost all ML algorithms is *training*, which refers to the process where we fit the model to our data by adjusting the model parameters. A central part of the training is data, which is typically divided into three sets. The majority of the data (typically 60-80%) is used to optimize the model and is called the *training set*. Some of the data (10-20%) is used as a *validation set*, which we can use to see how well the model is able to generalize and work on unseen data. This set can also be used to adjust *hyperparameters*, which are the parameters that the programmer manually sets (i.e. not learned by the model/algorithm). Lastly, the final part of the data is used to create a *test set* (10-20%), which is used to compare the performances of different models against each other.

2.2.1 Transfer Learning

Huge language models, such as BERT, are based on theory from *transfer learning*. Transfer learning means that we develop a model to solve a problem, and then use the knowledge gained to solve a different, related problem (Goodfellow et al. 2016, p. 534). This mimics the way humans learn, as we are able to generalize knowledge well. For example, knowing how to drive a car makes it much easier to learn how to ride a motorcycle. Analogously, it is much easier to train a machine learning model to detect cats in an image, if it is already good at detecting dogs.

For language models, this means dividing the training phase into two parts – *pre-training* and *fine-tuning*. In the pre-training phase, the model is trained on large amounts of data, with the goal of it being good at understanding natural language, such as grammar or the semantic meaning of words and sentences. Using the pre-trained model, we can fine-tune it with domain-specific data to solve a wide range of language-related problems. Using this approach, the training times are drastically reduced, as fine-tuning is usually quick. For BERT, pre-training took Google 4 days, while fine-tuning the model usually takes just a few hours (Devlin et al. 2019).

2.2.2 Feedforward Neural Networks

The machine learning models we are working with in this thesis are *artificial neural networks*. They come in many variants and together they make up the branch of machine learning called *deep learning*. They are used to mimic the way the human brain learns, with neurons that are connected. We begin by going into feedforward neural networks (FFNN), which are the vanilla neural networks. Often, FFNNs are used together with other types of neural networks to create a larger model (Goodfellow et al. 2016, p. 164).

Model Architecture

The basic building blocks of a neural network are the neurons, which contain a numerical value called an *activation*. They are ordered in a layered structure, where every neuron in each layer is connected to every neuron on the next layer (except for the last layer). This way, the information flows from the first to the last layer, hence the name feedforward neural network. There are no connections between neurons within a layer.

There are three types of layers – input, hidden and output layer. We can have many hidden layers, and the number of layers in the network is called the network's *depth*. The connections between the neurons are called *weights*, which is a numerical value that tells how strongly connected those neurons are. Also associated with a neuron is a *bias*, which is a value that tells us how impactful that neuron is in the entire network. Below is a graphical visualization of the network.

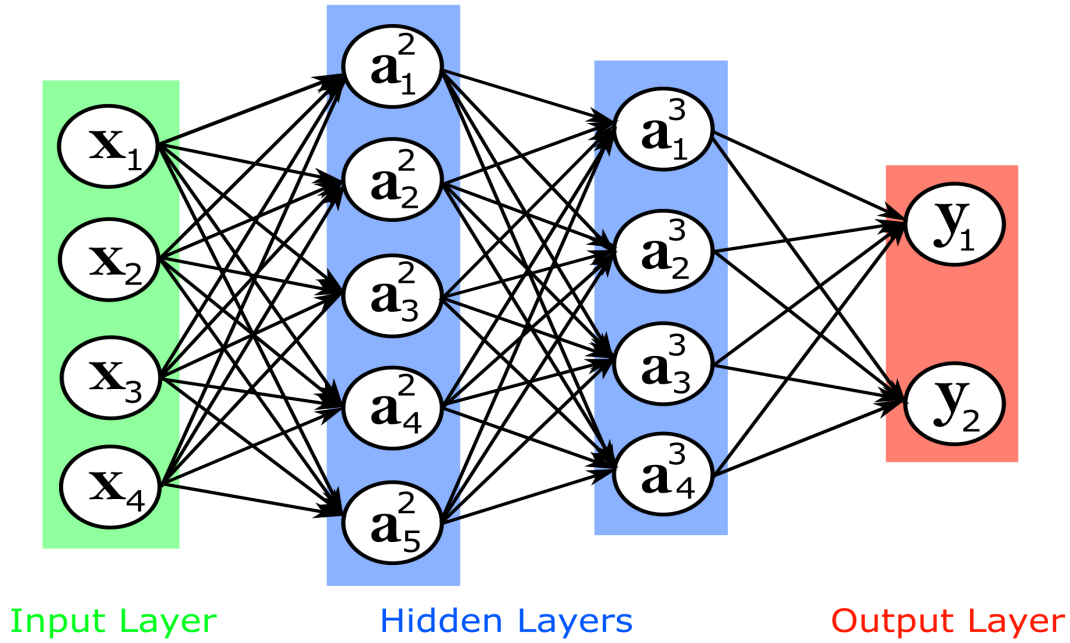


Figure 2.4: The architecture of a feedforward neural network

Mathematically, the FFNN is nothing more than a function F that takes in a vector $\mathbf{x}$ and returns a vector $\mathbf{y} = F(\mathbf{x})$. To understand how the output is calculated, we introduce the following notation:

- $l = 1, 2, \ldots, L$ denotes the different layers in the model, each containing n_l neurons
- w^l_{jk} : weight for the connection from the k-th neuron in layer $l-1$ to the j-th neuron in layer l
  $W^l \in \mathbb{R}^{n_l \times n_{l-1}}$ is a matrix for all weights in layer $l = 2, \ldots, L$
- b^l_j : bias of the j-th neuron in layer l
  $b^l \in \mathbb{R}^{n_l}$ is vector for all biases in layer $l = 2, \ldots, L$
- a^l are the activations for all the nodes in layer l. Here we let $a^1 = \mathbf{x}$ and $a^L = \mathbf{y}$

The activation of a neuron is based on all the neurons in the previous layer in the following way

$$a^l_j = \sigma_l\left(\sum_{k=1}^{n_{l-1}} w^l_{jk} a^{l-1}_k + b^l_j\right) \Rightarrow a^l = \sigma_l\left(W^l a^{l-1} + b^l\right). \tag{2.9}$$

The function $\sigma_l(\cdot)$ is known as an *activation function*, which acts elementwise. Its purpose is to introduce nonlinearity to the model, which makes the model more flexible towards nonlinear data. Without it, the network is just a series of affine transformations. For the hidden layers the ReLU (Rectified Linear Unit) is popular, given as

$$\sigma(\mathbf{x}_i) = \max(0, \mathbf{x}_i). \tag{2.10}$$

Another very important activation function is the softmax function

$$\sigma(\mathbf{x}_i) = \frac{e^{\mathbf{x}_i}}{\sum_{j=1}^{n} e^{\mathbf{x}_j}} \in [0, 1]. \tag{2.11}$$

It is very useful, because $\sum_{j=1}^{n} \sigma(\mathbf{x}_j) = 1$, which means that the output can be viewed as a probability distribution. It is therefore normal to use it as the activation function for the output layer when doing classification.

Putting everything together, the entire neural network can be summed up in one equation

$$F(\mathbf{x}) = \sigma_L\left(W^L \sigma_{L-1}\left(W^{L-1}\sigma_{L-2}\left(\ldots\sigma_2\left(W^2\mathbf{x} + b^2\right)\ldots\right) + b^{L-1}\right) + b^L\right) = \mathbf{y}. \tag{2.12}$$

Training a Neural Network

The training of a neural network is done by solving a minimization problem, where the weights and biases are the parameters we are optimizing. The function we are minimizing is called a *loss function* (also called objective or cost function), which we denote L. A simple loss function used in regression is the mean squared error (MSE), which for training samples $\mathbf{x}_i$, $i = 1, 2, \ldots, n$ and correct values $\mathbf{y}_i$, $i = 1, 2, \ldots, n$ is given as

$$L_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} \|F(\mathbf{x}_i) - \mathbf{y}_i\|_2^2. \tag{2.13}$$

If we are doing classification, the *cross-entropy loss* is often used. It requires that the output of the network forms a probability distribution, hence it is often used in conjunction with the softmax function. If the number of classes is C, let p_j denote the predicted probability that a sample belongs to class j and $y_j \in \{0, 1\}$ be the ground truth label. Then the cross-entropy loss for a single sample takes the following form

$$L_{\text{CE}} = -\sum_{j=1}^{C} y_j \log(p_j), \tag{2.14}$$

in which we average the loss over all training samples.

To solve the optimization problem, we use *gradient descent*, an iterative minimization technique (Goodfellow et al. 2016, p. 80-84). The idea behind the method is simple – calculate the gradient of the loss function at the point we are currently at, which is the direction of steepest ascent, and move a small step in the opposite direction. Mathematically, we write this as

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \mu_k \nabla_{\boldsymbol{\theta}} L, \tag{2.15}$$

where $\boldsymbol{\theta}_k$ represents all the weights and biases at time step k. The calculation of the gradient is done by a technique called backpropagation. We do not go into the details, but put simply it involves calculating the gradient at the end of the network and propagating the derivatives of the parameters backward using the chain rule.

Above, μ is the *learning rate*, a positive scalar determining the step length. A too large learning rate will often result in us overshooting the minimum and the loss increasing. In contrast, choosing it too low will result in slow convergence. The learning rate is therefore often shortened as the training progresses, to find a balance.

Trying to take every training sample into account when doing gradient descent can be very time consuming. Instead, we use *stochastic gradient descent* (SGD), in which we randomly divide the training set into smaller batches, and approximate the gradient as an average of the samples in each batch. The number of samples in each batch is called the *batch size* and an iteration over all the batches is called an *epoch*, both of which are hyperparameters.

2.2.3 Recurrent Neural Networks

Most of the data we are working with is textual data, like sentences or documents. Recurrent neural networks (RNN) are a family of neural networks that are used for handling sequential data as input. They were the forerunners for the models we use in this thesis, and by understanding the basics of RNNs, we are better able to understand the motivation behind the Transformer framework, which is presented later.

The difference between a normal FFNN and an RNN is the use of *recurrence*. In an RNN, the hidden state depends on the hidden state calculated before. The hidden state is just the activations of all the hidden layers, denoted $\mathbf{h}_t$ for time step t. In other words, the activation of a neuron is fed back to itself in each iteration. If we denote the input as a sequence of vectors $\{\mathbf{x}_t\}_{t=1}^{T} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$, we can express the recurrence with the following relation:

$$\mathbf{h}_t = \sigma(\mathbf{h}_{t-1}, \mathbf{x}_t; \boldsymbol{\theta}). \tag{2.16}$$

After having processed the entire sequence, the final hidden state can be viewed as a summary of the entire sequence (Goodfellow et al. 2016, p. 367-374). Figure 2.5 shows how a simple RNN with one hidden layer looks like.

A considerable downside with RNNs is that they suffer from *short-term memory*. As we process a sequence, the hidden state is predominantly affected by the recently processed elements, while the earliest elements become "forgotten". This is a result of vanishing or exploding gradients, a common problem within deep learning. Trying to combat this issue, long short-term memory (LSTM) and gated recurrent units (GRU) were created, which we will not go further into.

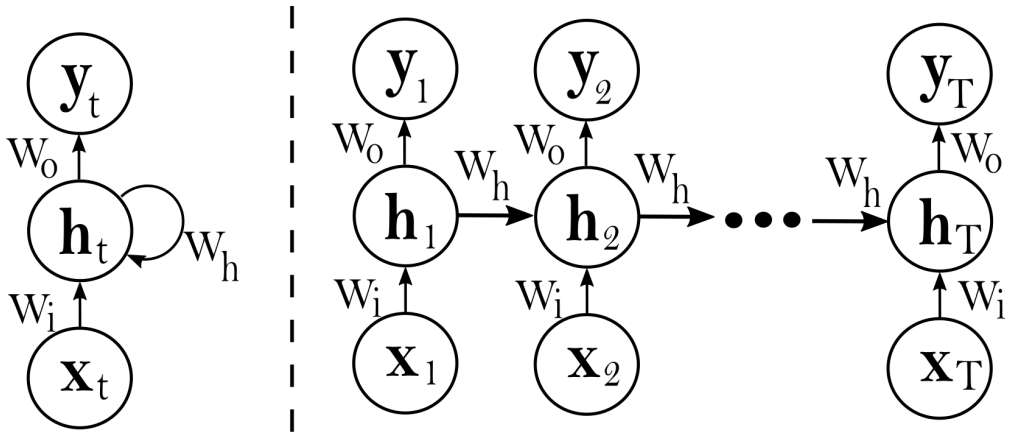


Figure 2.5: A simple RNN with one hidden layer. The circles represent layers, and the Ws are weight matrices. On the left is the compact recurrent form, and on the right is the unfolded representation.

2.3 Natural Language Processing

Natural language processing (NLP) is a subfield within computer science that revolves around computers being able to understand human language. It is the intersection between linguistics and artificial intelligence and has made huge progress with the introduction of neural networks. The subject also includes language models (LM), which are models that predict the probability distribution to a sequence of words. This can be used to solve many challenges within NLP, like machine translation and speech recognition (Eisenstein 2018, p. 1, 125). BERT is an example of such a language model, which means that these topics are central in this thesis.

2.3.1 WordPiece Tokenization

BERT uses a technique called WordPiece tokenization to create a vocabulary of possible inputs (Wu et al. 2016). Tokenization is the process of chopping up sentences into a sequence of smaller units called *tokens*. Tokens are usually words, but can also be smaller pieces of text that are meaningful for processing (Manning et al. 2009, p. 22).

In the WordPiece model, a vocabulary is created by breaking some words into sub-words, which can be syllables or individual characters. This is primarily done on more uncommon words, and the vocabulary therefore contains both entire words and sub-words. Below is an example that illustrates how this may look.

- **Input Sentence:** Example of WordPiece tokenization
- **List of Tokens:** [example, of , word , ##piece , token , ##ization]

The ## symbol tells us that the token is not at the start of a word, which means we can recover the original sentence without ambiguity.

The construction of the vocabulary is entirely data-driven using a greedy algorithm. Initially, the vocabulary only contains individual characters. After looking at a corpus of text, the model finds which merging of two tokens is most likely to appear in the corpus. We continue this process until the vocabulary has reached its desired, pre-specified size.

Two major benefits of using WordPiece tokenization are that you can restrict the vocabulary to a reasonable size and we don't run into the problem of processing words we haven't seen before, because we can decompose them into known sub-words.

2.3.2 Word Embedding

A key challenge within NLP is to extract meaning from text. It is difficult for a computer to know what a word means and how it relates to its context, just by looking at the sequence of characters it is made up from. This is where we can make use of word embedding.

Word embedding means that we take a word and *embed* it into a high-dimensional vector space of pre-defined size. The embedding space usually has a few hundred dimensions, but this may vary. The goal is that words with similar meanings also have vectors that are similar. Having the words represented as vectors allow the computer to use mathematical operations to show relationships between words, like addition or dot product. However, more importantly, we can use the vectors as a part of a neural network, which is what we are going to need them for later. The size of the embedding space is also much smaller than the number of words in the dictionary, meaning that we drastically reduce the number of dimensions, and therefore also reducing the computation times (Eisenstein 2018, p. 327-341).

There exist many different ways to create word embeddings, but nowadays neural networks have become increasingly more popular. To explain how these models work, we can use *word2vec* as an example (Mikolov et al. 2013). In the word2vec model, the creators propose two shallow neural networks, called Continuous-Bag-of-Words (CBOW) and Skip-gram, to learn these embeddings. In CBOW, we are given a sentence where the middle word is removed, and the goal is to predict what this word is. Skip-gram works the other way, where we are given one word, trying to predict what the surrounding words are. The input words are represented as one-hot encoded vectors, and the networks only have one hidden layer. The structures of these networks are shown in figure 2.6. After training the networks on huge amounts of text, the weights in the hidden layer represent the word embeddings, where each row in the weight matrix corresponds to one word in the dictionary. In both CBOW and Skip-gram we make use of the context the words are in, and the model is therefore able to understand meaningful relationships between words. A famous example that showcases this, is that when you calculate $vec("king") - vec("man") + vec("woman")$, the resulting vector is very close to $vec("queen")$.

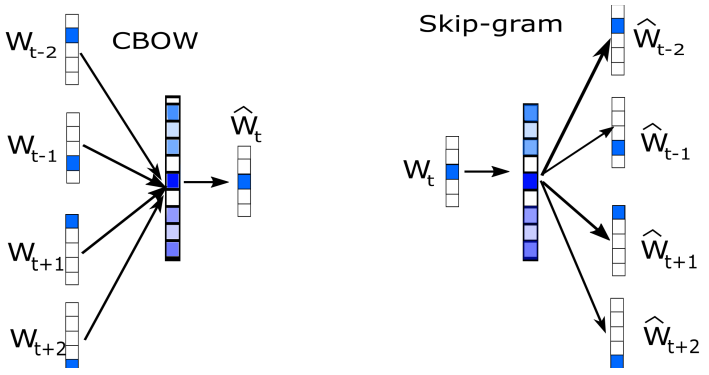


Figure 2.6: The neural networks used to learn word embeddings in the word2vec model.

2.3.3 Sequence-to-sequence Models

Many neural language models belong to the class of sequence-to-sequence models (seq2seq). Seq2seq models are machine learning models that use an input sequence $\{\mathbf{x}_t\}_{t=1}^{T_x}$ and generates a sequence $\{\mathbf{y}_t\}_{t=1}^{T_y}$ as output (Sutskever et al. 2014). In this context, we only consider sequences of words.

Common for many of these models is the *encoder-decoder framework*, as proposed by Cho et al. (2014), illustrated in figure 2.7. By splitting the model into two parts, we allow the input and output sequences to have different lengths. Both the encoder and decoder are RNNs (or LSTMs), and are trained jointly.

The first half of the framework is the encoder. It uses the input sequence to create a single, fixed-length vector $\mathbf{c}$, called the *context*. It is given by $\mathbf{c} = \mathbf{h}^e_{T_x}$, which is a summary of the input sequence. Here, $\mathbf{h}^e_t$ is the hidden state at time step t in the encoder (and likewise $\mathbf{h}^d_t$ for the decoder).

The context vector is then used by the decoder to generate the output sequence. We predict each output sequence element based on a conditional probability given the previously generated elements and the context. We write this as

$$P\big(\mathbf{y}_t|\mathbf{y}_1,\mathbf{y}_2,\ldots,\mathbf{y}_{t-1},\mathbf{c}\big) = g\big(\mathbf{h}^d_t,\mathbf{y}_{t-1},\mathbf{c}\big) \tag{2.17}$$

$$\mathbf{h}^d_t = f\big(\mathbf{h}^d_{t-1},\mathbf{y}_{t-1},\mathbf{c}\big). \tag{2.18}$$

Here, f is an activation function and g is a function that produces a probability distribution over all possible sequence elements, typically the softmax function. We choose the element with the highest probability and stop generating new elements when the end-of-sequence symbol is generated (Cho et al. 2014).

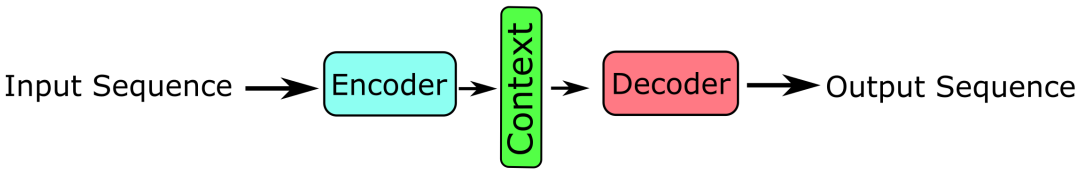


Figure 2.7: Encoder-Decoder Architecture

2.3.4 Attention

A large drawback with the seq2seq model as described above is its ability to deal with long sequences. By using RNNs, we run into the problem of short-term memory, even though this could be mitigated by using LSTMs or GRUs. The elements that were processed last affect the context the most, which is is problematic if the last elements are not relevant. The context is also not able to pick up relations between the first and the last elements. In addition, trying to squeeze all the information about a long sequence into a single vector is often not sufficient to be able to predict a good output sequence.

To fix this issue, the *attention mechanism* was added to the model, first proposed by Bahdanau et al. (2015). Instead of feeding only the last hidden state to the decoder, we give it *all* the hidden states that were produced by the encoder. The idea is that $\mathbf{h}_t^e$ corresponds to $\mathbf{x}_t$, which means that all input elements are equally important. Using this idea, the short-term memory issue is removed.

With the new approach, a new context vector $\mathbf{c}_t$ is created for each time step t in the decoder. The context vector is now calculated as a weighted sum of all the received hidden states

$$\mathbf{c}_t = \sum_{s=1}^{T_x} \alpha_{ts} \mathbf{h}_s^e \tag{2.19}$$

The weights are calculated as follows

$$\alpha_{ts} = \frac{\exp\left(e_{ts}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{tk}\right)}, \quad e_{ts} = a\left(\mathbf{h}_{t-1}^d, \mathbf{h}_s^e\right). \tag{2.20}$$

a is an *alignment model*, i.e. a model that estimates how much output elements around position t match input elements around position s. The creators, Bahdanau et al. (2015), use a feed-forward neural network as their alignment model, which is trained together with the rest of the model. After having calculated $\mathbf{c}_t$, $\mathbf{h}_t^d$ and $\mathbf{c}_t$ is concatenated and fed into a final FFNN to predict output element $\mathbf{y}_t$.

The alignment model is a key concept behind the attention mechanism. By learning to align output elements with input elements, the decoder can decide which elements it wants to "pay attention to", for each output element it predicts. Additionally, by using all the encoder's hidden states, more of the information from the input sequence is available to the decoder, compared to compressing all the information into a single context vector.

2.4 The Transfomer

Based on the concept of attention, the Transformer was created by Vaswani et al. (2017), first presented in their famous research paper "Attention Is All You Need". A lot of the recent language models, like BERT for instance, are based on ideas and innovations presented in that paper. Transformer-based models have rapidly become dominant within NLP (Wolf et al. 2020), and we have therefore devoted an entire section to the Transformer. We do not go into all the details about the transformer model here, but present some of the innovations from the paper that are useful.

2.4.1 Self-Attention

The Transformer, like many other seq2seq models, consists of encoders and decoders, but instead of using RNNs to extract information, they rely on *self-attention*. Self-attention is similar to the attention mechanism already discussed, but instead of aligning elements of two different sequences, we align a sequence with itself, hence the name. Using this idea, we can, for example, connect words in a sentence to each other, without caring about where in the sentence the words appear. Vaswani et al. (2017) showed that this concept on its own was powerful enough to be useful, thus explaining the name of the research paper.

Instead of using attention between the encoder and decoder, self-attention is a built-in layer of the encoder and decoder. The Transformer consists of several encoders and decoders stacked on top of each other, thus extracting more information about the sequences. Each decoder receives data from the last encoder, and the first decoder also receives the parts of the output sequence already generated. An overview of the Transformer architecture is shown below.

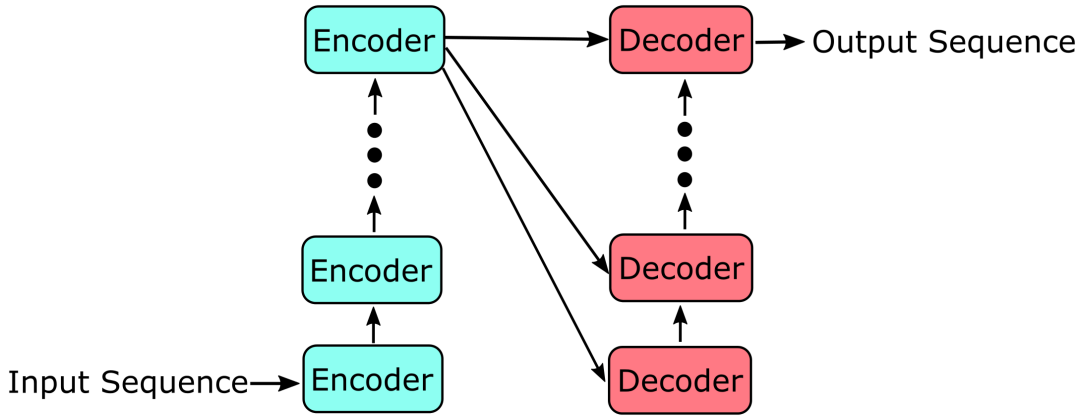


Figure 2.8: The Transformer Architecture

To fully understand how self-attention works, we need to dive into the underlying mathematics. As input to the self-attention layer, we use a sequence $\{\mathbf{x}_t\}_{t=1}^{T} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$, $\mathbf{x}_t \in \mathbb{R}^{d_m}$ (not the same as the input sequence to the model). For each element $\mathbf{x}_t$, we calculate a query vector (not to be confused with a query in an IR setting) $\mathbf{q}_t \in \mathbb{R}^{d_k}$, a key vector $\mathbf{k}_t \in \mathbb{R}^{d_k}$ and a value vector $\mathbf{v}_t \in \mathbb{R}^{d_v}$. These vectors are found by linearly transforming the input as follows

$$\mathbf{q}_t = W^Q \mathbf{x}_t, \quad W^Q \in \mathbb{R}^{d_k \times d_m} \tag{2.21}$$

$$\mathbf{k}_t = W^K \mathbf{x}_t, \quad W^K \in \mathbb{R}^{d_k \times d_m} \tag{2.22}$$

$$\mathbf{v}_t = W^V \mathbf{x}_t, \quad W^V \in \mathbb{R}^{d_v, \times d_m} \tag{2.23}$$

where the weight matrices W^Q, W^K and W^V are learned.

Using these vectors, we are able to calculate the output of the self-attention layer. The output is given as

$$\mathbf{z}_t = \sum_{s=1}^{T} \alpha_{ts} \mathbf{v}_t, \ t = 1, 2, \ldots, T \tag{2.24}$$

where

$$\alpha_{ts} = \frac{\exp(e_{ts})}{\sum_{i=1}^{T} \exp(e_{ti})}, \quad e_{ts} = \frac{\mathbf{q}_t^\top \mathbf{k}_s}{\sqrt{d_k}} \tag{2.25}$$

We see that this is analogous to 2.20, where we now use a scaled dot product as alignment model. The scaling factor $\frac{1}{\sqrt{d_k}}$ is there to prevent the dot product from growing too large in magnitude, i.e. pushing the softmax function into regions where its gradient is small and the learning is slower.

In practice, all the outputs are computed simultaneously by making use of matrices. By letting the query, key and value vectors be the rows of matrices Q, K and V, respectively, we can compute the output of a self-attention layer in the following compact form

$$\text{Self-Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)V. \tag{2.26}$$

2.4.2 Multi-Head Attention

Using self-attention, the model is able to understand relevance between different elements of a sequence. However, there may be many types of relevance relations between elements, and to address these relations, the Transformer makes use of *multi-head attention*.

Instead of only projecting the input to a single instance of query, key and value vectors, multi-head attention does this h times. Each instance of self-attention is called an *attention head*, and have their own learned weight matrices, W_i^Q, W_i^K, W_i^V, $i = 1, 2, \ldots, h$, all initialized randomly. After calculating self-attention h times in parallel, the outputs are concatenated and projected using a learned matrix $W^O \in \mathbb{R}^{hd_v \times d_m}$. We write this as

$$\text{MultiHead}\big(\{\mathbf{x}\}_{t=1}^T\big) = \text{Concat}\big(\text{head}_1, \ldots, \text{head}_h\big)W^O, \tag{2.27}$$

$$\text{where head}_i = \text{Self-Attention}\big(W_i^Q X^\top, W_i^K X^\top, W_I^O X^\top\big). \tag{2.28}$$

Above $X = \big[\mathbf{x}_1, \ldots, \mathbf{x}_T\big]^\top$, i.e. a matrix where the rows are the input sequence vectors. By using several attention heads, the model now has multiple representation subspaces to embed meaning into. It also increases the number of parameters in the model, making it bigger and more flexible.

2.4.3 Positional Encoding

In the multihead-attention mechanism described above, the order of the sequence elements is irrelevant and hence the model does not have any sense of position. However, the order may be of importance and therefore more information could be extracted. To fix this, the Transformer makes use of *positional encodings*.

Positional encodings are added to the input or the already generated output sequence, before they enter the first encoder or decoder. To positional encoding is given by

$$PE(t,i) = \begin{cases} sin\big(t/10000^{2i/d_m}\big) & \text{if } i \text{ even} \\ cos\big(t/10000^{2i/d_m}\big) & \text{if } i \text{ odd.} \end{cases} \tag{2.29}$$

t is the position of the element in the input sequence, and i refers to the i-th element in the vector. By using this combination of sines and cosines, we can calculate $PE(t+k, i)$ as a linear function of $PE(t, i)$ for a fixed k. As a result, the creators hypothesize that the model easily learns relative positions, which is exactly what we want.

2.4.4 Layer Normalization

To speed up training, the model makes use of layer normalization, introduced by Ba et al. (2016). This means that for each hidden layer in the model, we normalize each activation by subtracting the mean and divide by the standard deviation of the activations in the layer. This is done on both the self-attention layers and the hidden layers of the FFNNs.

Layer normalization helps speed up training in two ways. First, since the distributions of activations now are very similar for each training sample, it becomes easier to optimize the weights in the model. Secondly, it helps to deal with the vanishing/exploding gradient problem, because it becomes easier to tune the hyperparameters, especially the learning rate, to avoid running into areas where the gradients vanish or explode.

2.4.5 A Game Changer within NLP

The Transformer has become a game changer within the world of NLP, outperforming its predecessors on a wide range of language-related tasks (Wolf et al. 2020). Since the Transformer does not rely on any form of recurrence, a much larger amount of the computations to be carried out in *parallel*, vastly reducing the time it takes to train the model. Combining this with its ability to handle long sequences due to the self-attention mechanism, means that it deals with the two major problems of RNN-based models, namely slow training times and short-term memory.

Reducing the time it takes to train the model means that one can increase the number of parameters in the model, further increasing its performance. This has laid the foundation for bigger and better models, like Google's BERT and OpenAI's GPT model. Since these models are open-source, they have rapidly become the most popular language models to use (Wolf et al. 2020).

2.5 BERT

One year after the Transformer was published, a new language model called BERT was presented by Google and their team lead by Devlin et al. (2019). BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers, and uses many of the innovations the Transformer brought to the table. Later, several other models have been built based on BERT, which we briefly look at in the end.

In this section, we present the pre-trained BERT model, showing its architecture and how it was pre-trained. The theory presented so far should equip us with all the knowledge and tools to fully understand the model. This model can be modified to solve a wide range of language-related tasks, by adding customized output layers on top. The pre-trained model is the foundation of a lot of the subsequent work in this thesis and it is the culmination of the entire background theory chapter.

2.5.1 Model Architecture

Figure 2.9 shows the architecture of the pre-trained BERT model. BERT is available in different sizes, where the two described in the paper are BERT-Base and BERT-Large. The architecture of these are identical, and the only difference is the size and number of parameters. All the sizes, dimensions and number of parameters for the two main BERT models are presented later.

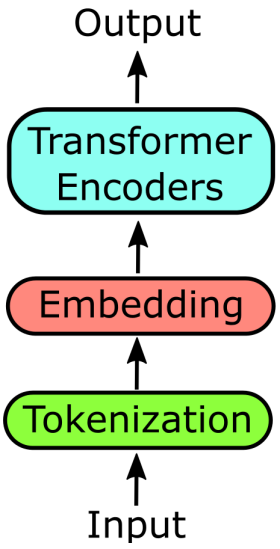


Figure 2.9: The BERT Architecture

Input, Tokenization and Embedding

BERT can use both one or two sentences as input, where a sentence refers to a contiguous span of text, and not necessarily a linguistic sentence only. We refer to the first sentence as A and the second as B. Having the option of using one or two sentences, BERT can be used to solve a wider range of language tasks. This is especially useful for us, trying to use BERT for document ranking.

The input sentences are divided into a sequence of tokens using WordPiece tokenization with a vocabulary containing 30000 tokens. Some of the tokens have a special purpose and need to be further explained:

- [CLS] - used to do sequence classification tasks
- [SEP] - used to separate the two input sequences (if there are two)
- [PAD] - if the input sentences are shorter than the maximum length, we use this token to fill out the remaining sequence elements
- [UNK] - used if we find text that is not in the vocabulary, for example, a special character

All the tokens in the vocabulary have a learned embedding, each with d_m dimensions. Added to the token embedding vectors are learned positional encodings and segment embeddings, which correspond to the tokens belonging to the first or second sentence.

The final result from the tokenization and embedding process, is a sequence of fixed length $\mathcal{X} = (\mathbf{x}_0, \ldots, \mathbf{x}_{T-1})$, usually of size 128, 256 or 512 (max).

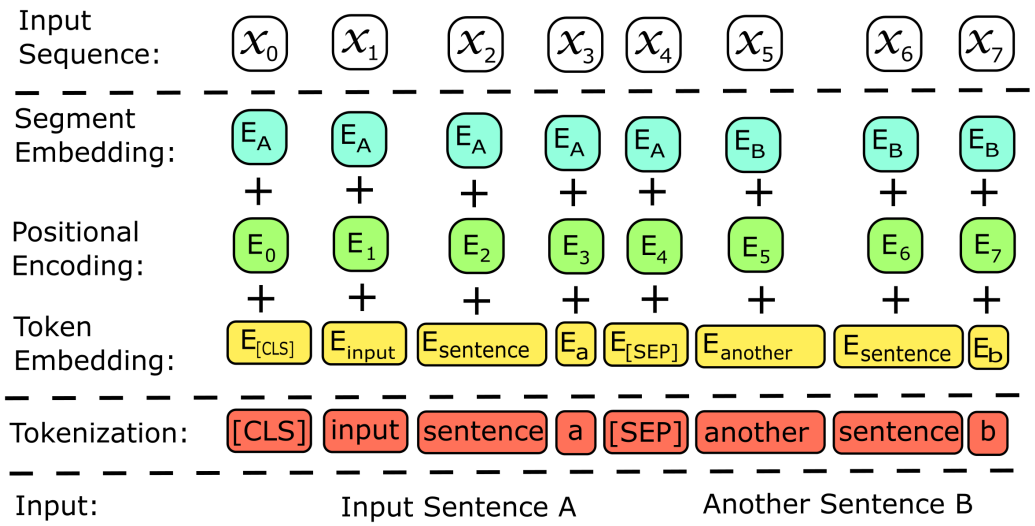


Figure 2.10: Tokenization and Embedding in BERT

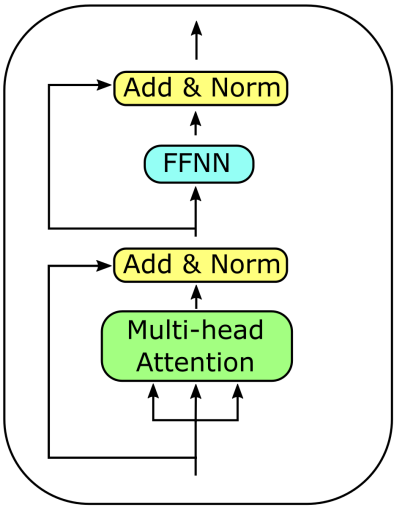


Figure 2.11: The Transformer Encoder Used in BERT

The Transformer Encoders

The transformer encoders are the most important part of the BERT model, which are entirely based on the work of Vaswani et al. (2017). BERT uses several of these stacked on top of each other, allowing the model to extract a lot of information about the input sequence. Every encoder is identical in size. Figure 2.11 shows the components of one such encoder.

First, the input sequence of the encoder is fed through a multi-head attention layer, as described earlier. The dimensions in the query, key and value matrices are equal, denoted d_k, and the number of attention heads is chosen such that the output of the attention layer has the same size as the input.

Residual connections are used in the encoder, which means that the input and output of the layer are added. This allows the model to retain the original information longer, including the positional encodings. After the addition, layer normalization is applied. These actions are illustrated by the "Add & Norm" block in the figure.

The information gained in the attention layer is pushed through a small FFNN, which is there to process the attention vector into a form that is easier to handle by the next encoder. The attention vectors are processed one at a time using the same weights, which means that these computations can be parallelized. The FFNN consists of two linear transformations with a ReLU activation function in between:

$$FFNN(\mathbf{x}) = \max(W_1\mathbf{x} + b_1, 0)\,W_2 + b_2, \tag{2.30}$$

where W_1, W_2, b_1 and b_2 are learned weights and biases.

Specification	BERT-Base	BERT-Large
#Transformer Encoders (L)	12	24
#Attention Heads (A)	12	16
Length of Sequence Vectors (H)	768	1024
Max Sequence Length	512	512
Total Parameters	110M	340M

Table 2.1: BERT Size and Specifications

The output of the last encoder is the output of the pre-trained BERT model. That means that the output is a sequence of vectors with d_m elements, one for each token in the input sequence $\mathcal{X}$. This output can be used as it is or used in a modified, fine-tuned model.

Table 2.1 shows the size of the two main BERT models, BERT-Base and BERT-Large. The letters in parentheses denote the commonly used abbreviation. As we see, the numbers of parameters are huge, and the creators found that bigger models yield better results (Devlin et al. 2019).

2.5.2 BERT Pre-Training

During pre-training, BERT is taught to solve two simple but powerful tasks. These tasks have been chosen to make the model understand language at both token level and sentence level, and they are done at the same time.

Masked Language Modeling

To learn relationships between words, the creators introduce Masked Language Modeling (MLM). In the input sequence, some of the tokens have been masked, and the goal is to predict what these were, which is also known as the Cloze task. To predict words, an extra linear layer followed by a softmax function has been added to the output of the last encoder. This creates a probability distribution over all the tokens in the vocabulary, which we can use to predict. Only the masked tokens are used to calculate the loss of the prediction.

For every training sample, 15% of the tokens are being masked at random. The tokens are changed according to the following distribution:

- 80% are changed to [MASK] - a special token that represents a masked token
- 10 % are changed to a random token, where more common tokens are chosen more often
- 10 % stays unchanged

The idea behind not using the [MASK] token every time, is because this token never appears during fine-tuning, so we do not want to bias the model towards this token.

By using MLM, we train the model *bidirectionally*, which means that we give it information both to the left and right of the masked tokens. Training the model this way is slower than pure left-to-right or right-to-left models, but the increased performance outweighs this.

Next Sentence Prediction

We also want the model to be good at understanding relationships between sentences, which is why BERT is taught Next Sentence Prediction (NSP). NSP is a binary classification task where we are given two sentences, where the goal is to tell if the second one actually comes after the first one or not. This is true in 50% of the training samples, and we use the [CLS] token to base our prediction on.

Combining NSP and MLM, we can train a model to be good at understanding natural language. The tasks themselves are not useful for real-life applications, but they require sophisticated language understanding to solve. They are also hard to do, which means that the model is always able to improve during training.

The tasks' simplicity also allows the learning to be *self-supervised*. This means that we teach the model to do a supervised task, without labeling the data ourselves. Instead, the data is labeled by the computer during the training process. Consequently, we can use huge amounts of unlabeled data, and BERT uses all of the English Wikipedia combined, adding up to 2500M words, in addition to using BooksCorpus containing 800M words. Having this amount of training data is vital in the creation of a powerful, big language model. It took Google four days on clusters of TPUs to train each of the two BERT models. However, having the pre-trained model as a starting point, we drastically reduce the time it takes to fine-tune the model.

2.5.3 Other BERT-Based Models

In the wake of the release of BERT, several other models were created that aimed to improve the BERT architecture and pre-training. Together, they highlight some of the improvement points that BERT has and help to progress the field of NLP and language models. Here, we shortly present some of these models, listed in chronological order of their creation.

RoBERTa

RoBERTa, which stands for Robustly Optimized BERT Approach, was Facebook AI's, together with the University of Washington, attempt at improving the way BERT was being pre-trained. By analyzing the different components of the pre-training procedure, they created their own optimized approach. The model was compared to BERT on three language tasks – question answering, sentence classification and reading comprehension (Liu et al. 2019).

A significant improvement was found using more data. BERT was trained on 16 GB of text, while RoBERTa was trained on 160 GB. Additionally, training the model for more steps, increasing the number from 100K to 500K, also resulted in increased performance.

The pre-training tasks used in BERT were also modified. In the Masked Language Modeling task, the masking was originally done once as a part of data preprocessing. RoBERTa, on the other hand, adopts a dynamic masking model, which means masking sequences right before they are fed into the model. This way, the data is augmented to create more training samples. Secondly, they found that removing the Next Sentence Prediction task, slightly improved performance.

ALBERT

Huge language models like BERT have shown that increasing the number of parameters leads to improved performance. However, with several hundreds of millions or even billions of parameters, the threshold for how big models can be is close to being reached, both in terms of computation time and memory limitations. ALBERT, meaning A Lite BERT, tries to attack this issue by proposing two parameter reduction techniques (Lan et al. 2019).

The first method is to divide the embedding step into two parts. In BERT, the embedding of the sequence tokens is done in one step, which means using a matrix with dimensions $V \times H$, where V is the vocabulary size, usually around 30000, and H is the length of the sequence vectors, usually 768. This accounts for a lot of the total parameters in the model. ALBERT on the other hand, first embeds the tokens into intermediate vectors of size E, before transforming them into vectors of size H. This results in two matrices with dimensions $V \times E$ and $E \times H$, which in the case of $E << H$, greatly reduces the number of parameters in the model. Additionally, if we want to increase H, the number of parameters is not significantly increased.

The other thing they did, was to use cross-layer parameter sharing. This refers to using the same parameter weights in all of the transformer encoders, both in the attention layers and the FFNN layers.

The reduction of parameters leads to a slight decrease in performance, naturally. However, an ALBERT configuration of BERT-Large has 18 times fewer parameters and training is 1.7 times faster, thus greatly improving parameter efficiency.

DistilBERT

Another model that tries to reduce the number of parameters and increase training and inference speed is DistilBERT (Sanh et al. 2019). It is based on the concept of *knowledge distillation*, which is a technique within machine learning based on a larger model, called the teacher model, transferring its knowledge to a smaller one, known as the student. This way, the smaller model can achieve better results compared to being trained on its own.

In practice, this is done by altering the loss function the student is trained to optimize. In the case of classification, a model is normally trained to minimize cross-entropy loss, as given by 2.14. In knowledge distillation, the ground truth labels are replaced by the probabilities estimated by the teacher, and is therefore called *soft loss*. Keeping the same notation as in 2.14, we write the loss function as

$$L_{soft} = -\sum_{j=1}^{C} \frac{t_j}{T} \log\Big(\frac{p_j}{T}\Big), \tag{2.31}$$

where t_j is the teacher model predictions. T is a parameter called *temperature*, which adjusts how soft the output distribution is. $T \to 0$ corresponds to the one-hot encoded case, while $T \to \infty$ corresponds to a uniform distribution. Often, setting $T = 1$ works well.

As a result of using the soft loss, the model becomes better at generalizations, instead of focusing on getting the one true class correct. If we consider the MLM task, for example, it is OK if we train the student to predict the word *rock*, even though the correct word was *stone*.

With DistilBERT, the creators managed to train a student model with BERT-Base as a teacher, which was 40% smaller and 60% faster, but still achieving 97% of the performance of its teacher.

Chapter 3

Related Work

This chapter presents some of the research that has currently been done on the area. Even though the exploration of the topic is in its early days, the great success BERT has achieved has captured the attention of many research groups. This has lead to great advancements in the performance of BERT-based ranking systems, but also motivates further research on unexplored topics.

In the chapter, we present some methods that demonstrate how BERT can rank documents, highlighting the differences in performance. We also introduce how BERT can be made both faster and better using task-specific knowledge distillation.

3.1 Using BERT for Text Ranking

Following the great results BERT had on many language-related tasks, it did not take long until people wanted to try it for ranking tasks within information retrieval. In this section, we present some of the models that show how BERT can be used for this purpose.

3.1.1 Beginning of the BERT Revolution

The first attempt to use BERT for text ranking was made by Nogueira & Cho (2019), only three months after BERT first was released. They used it for *passage ranking*, where the goal is to rank passages, which are paragraph-length extracts from longer texts such as books or web pages.

The idea behind their model was simple – estimate the probability of a passage being relevant, and use that directly to rank the passages. The model uses the query and the passage as the two input sentences in BERT. The estimate is done by calculating a score s, by attaching a simple fully connected layer on top of the output corresponding to the [CLS] token. This output is called the *passage representation* and denoted by $T_{[CLS]}$. s can then be calculated as follows

$$s = \text{sigmoid}\big(W^\top T_{[CLS]} + b\big), \quad W \in \mathbb{R}^{d_m}, b \in \mathbb{R}. \tag{3.1}$$

In other words, it is just a simple logistic regression model added to BERT. During training, we want to minimize the binary cross-entropy loss, which is given by

$$L = -\sum_{i \in J_{pos}} \ln(s_i) - \sum_{i \in J_{neg}} \ln(1 - s_i), \tag{3.2}$$

where J_{pos} and J_{neg} are the sets of indexes of the relevant and non-relevant passages respectively.

The model was evaluated on the MS MARCO dataset, making it comparable to many other existing models. The results from their experiments were impressive compared to their pre-BERT counterparts, which paved the way for a small revolution within text ranking

> Nogueira and Cho [2019] kicked off the "BERT revolution" for text ranking, and the research community quickly set forth to build on their results — addressing limitations and expanding the work in various ways. (...) The rest, as they say, is history (Lin et al. 2020, p.18).

3.1.2 Multi-Stage Rankers

Following the success of the simple model, people started trying out bigger and more advanced models. One such advancement was to use a multi-stage approach, which already existed for pre-BERT models. The basic idea behind is that several different BERT models are used in succession, allowing for increased inference. By shrinking the number of documents at each stage, bigger models with more expensive features can be used to rank the top-scoring documents (Lin et al. 2020, p. 68-70).

The first application of this approach using BERT was proposed by Nogueira et al. (2019). In their setup, they first use a pointwise model equal to the one described above, called monoBERT, followed by a pairwise model called duoBERT that re-ranks the list.

duoBERT considers two documents, D_i and D_j, at the same time and tries to estimate the probability $p_{i,j}$ that D_i is more relevant than D_j. The model is trained to minimize the loss given as

$$L_{\text{duo}} = -\sum_{i \in J_{pos}, j \in J_{neg}} \ln(p_{i,j}) - \sum_{i \in J_{neg}, j \in J_{pos}} \ln(1 - p_{i,j}), \tag{3.3}$$

where each pair of training documents is never both relevant or both not relevant.

To rank the documents, a relevance score for each document is calculated based on the pairwise probabilities. Five different aggregations are proposed:

- SUM: $s_i = \sum_{i \neq j} p_{i,j}$
- BINARY: $s_i = \sum_{i \neq j} \mathbb{1}_{p_{i,j} > 0.5}$
- MIN: $s_i = \min_{i \neq j} p_{i,j}$
- MAX: $s_i = \max_{i \neq j} p_{i,j}$
- SAMPLE: sample m documents without replacement, and sum the pairwise probabilities. This aims to decrease the number of inferences.

Results showed that using BINARY or SUM to do inference, increased the performance compared to just using monoBERT. On the other hand, it also means doing many more inferences, in which the tradeoff between speed and performance becomes relevant. If we consider a list containing k_0 and k_1 documents for the first and second ranking phase, respectively, we end up doing $k_0 + k_1(k_1 - 1)$ inferences. However, if k_0 and k_1 are chosen optimally, we could end up with a model that is both faster and better than compared to just using a single-stage ranker, due to the increased knowledge duoBERT brings (Lin et al. 2020, p. 72).

3.2 Ranking Documents with BERT

Based on the results from passage ranking, BERT was quickly adopted to be used for document ranking as well. However, using an entire document as input to BERT turned out to be a major challenge to overcome during this transition. BERT has a maximum input size of 512 tokens, and trying to keep the query, document and additional special tokens below this limit is not straightforward.

However, a lot of the knowledge gained from doing passage ranking can also be applied to document ranking. Most of the models that have been tried out so far are all based on the principle of splitting a document into shorter passages and combine results from each passage to infer the relevance of an entire document. Here, we present a couple of methods that use this technique in different ways. Results showing the performance of each model are displayed in table 3.1.

3.2.1 Passage Score Aggregation

Passage score aggregation was the first method to be tested for document ranking. The idea is to combine the relevance score for each query-passage pair, by using the same model as Nogueira & Cho (2019), in order to create a relevance score for the entire document. A simple approach was suggested by Dai & Callan (2019), which included the following key points:

- During training: Split the training documents into overlapping passages of 150 words with 75 words overlapping, and treat every passage from relevant documents as relevant, and vice versa. If the document has a title, add that to the beginning of every passage for additional information.
- During inference: Split the document in the same way and do inference on every passage. Aggregate the score for every passage to get the total document score. The aggregation can be done by either only considering the first passage score (denoted as BERT-FirstP), only the best passage score (BERT-MaxP) or the sum of all the scores (BERT-SumP).

The results from their experiments showed that taking the maximum passage score as the document score worked well, which is consistent with other similar techniques (Lin et al. 2020, p. 54-61). This shows that BERT is able to extract the most important meaning in a document based on only a small section of the document.

Extensions to this method have been tried, where the passages of a document are assigned different levels of relevance. This way, the model can learn better which passages that are the most important and value those more. However, the creators had to manually grade the relevance of each passage, which meant that they did not have enough training samples to make significant performance gains (Lin et al. 2020, p. 61). Nevertheless, the intuition behind this idea may have inspired the model in the subsequent section.

3.2.2 Passage Representation Aggregation

Passage Representation Aggregation (PARADE) was proposed by Li et al. (2020), with the goal of better relating passages to each other. Instead of aggregating the relevance score from each passage, they make use of the passage representations. They use these to create a document representation $D^{cls} \in \mathbb{R}^{d_m}$, where the relevance score for the query-document pair is calculated as a weighted sum over these elements, $s = W_d^\top D^{cls}$, where $W_d \in \mathbb{R}^{d_m}$ is learned.

Four different aggregation models are proposed to create the document representation:

- $\text{PARADE}_{\text{Avg}}$: Element-wise average pooling across passage representations:

$$D^{cls}[i] = \frac{1}{n} \sum_{k=1}^{n} T^k_{[CLS]}[i]$$

- $\text{PARADE}_{\text{Max}}$: Element-wise max pooling across passage representations:

$$D^{cls}[i] = \max_{k \in \{1,\dots,n\}} \left\{ T^k_{[CLS]}[i] \right\}$$

- $\text{PARADE}_{\text{Attn}}$: Weighted average across passage representations, which means that each passage contribute differently. This is done by applying a simple FFNN on top of the different passage representations:

$$w_1, \dots, w_n = \text{softmax}\left(W^\top T^1_{[CLS]}, \dots, W^\top T^n_{[CLS]}\right)$$
$$D^{cls} = \sum_{k=1}^{n} w_k T^k_{[CLS]},$$

  where $W \in \mathbb{R}^{d_m}$ is a learned weight.

- $\text{PARADE}_{\text{Transformer}}$ (also just called PARADE): Make use of two transformer encoders to extract information about the passages. The encoders are identical to those used in $\text{BERT}_{\text{BASE}}$ (section 2.5), and the input sequence vectors are the concatenation of the passage representations and a prepended [CLS] embedding. The output corresponding to the [CLS] embedding is used as the document representation. A graphical visualization is given in figure 3.1.

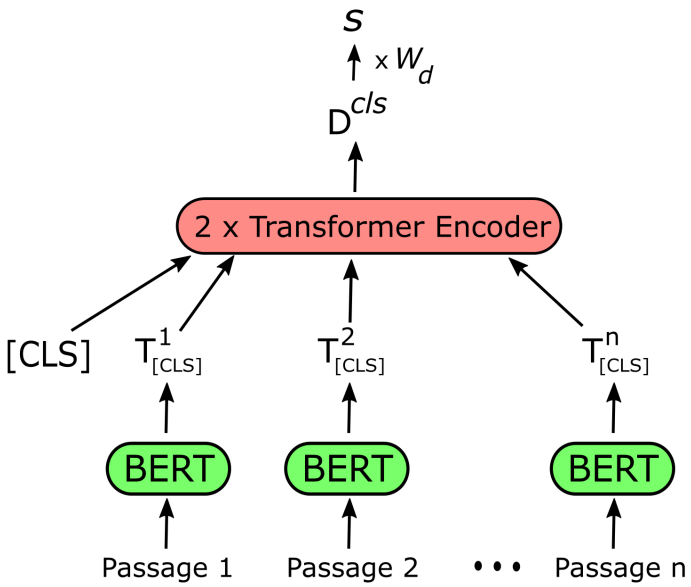


Figure 3.1: Architecture of PARADE. $T_{[CLS]}$ denotes a passage representation, D^{cls} is the document representation and s is the document relevance score.

Their results show that $\text{PARADE}_{\text{Avg}}$ performs worst, followed by $\text{PARADE}_{\text{Max}}$ and $\text{PARADE}_{\text{Attn}}$. However, PARADE clearly outperforms all of these, showing the usefulness of the transformer yet again. All the models except $\text{PARADE}_{\text{Avg}}$ significantly outperform models using passage score aggregation (Li et al. 2020).

There are a couple of advantages to making use of the passage representations that can help explain why this approach works better. Firstly, a lot more information is contained in the representations, whereas by only using the passage score, a lot of the information is lost. For instance, PARADE accounts for the order of the passages and how they all relate to make up the document. Secondly, the method allows the evaluation of a query-document pair during the training phase and inference to be identical, which also helps remove additional noise and sources of error.

Method	nDCG@20
BM25	0.424
BERT-FirstP	0.444
BERT-MaxP	0.469
BERT-SumP	0.467
$\text{PARADE}_{\text{Avg}}$	0.492
$\text{PARADE}_{\text{Max}}$	0.512
$\text{PARADE}_{\text{Attn}}$	0.513
PARADE	0.525

Table 3.1: Performance of BERT-based models on the Robust04 test collection.

3.3 Knowledge Distillation For Ranking

At the end of the last chapter, we briefly introduced the concept of knowledge distillation (KD) and how it could be used to create DistilBERT. In DistilBERT, the knowledge distillation was done in the pre-training phase, but KD can also be applied specifically to document ranking models. Several models have tried to use KD, and one such model is Simplified TinyBERT (STB), published by Chen et al. (2021) in March 2021.

The method is based on TinyBERT, which introduces a new way of doing KD that can be applied to many different tasks, not just document ranking (Jiao et al. 2019). Instead of the student model trying to mimic only the output predictions of the teacher, the student in TinyBERT is trained to mimic the output of *all* the layers. In practice, this is done by adding three extra losses to the soft loss function (eq. 2.31), corresponding to the embedding layer, the multi-head attention layers and the FFNN layers. We refer to the paper by Jiao et al. (2019) for details.

As the name indicates, Simplified TinyBERT applies a few simplifications to the way the student is taught. Firstly, whereas TinyBERT applies a general distillation to the pre-training as well, STD just initializes the weights of the student using the weights in the first layers of the teacher. Secondly, all the layers are distilled jointly in STB, whereas TinyBERT distills the intermediate layers and output layer separately. Lastly, in STB, the hard cross-entropy loss (eq. 2.14) is added to the total loss to help the model better distinguish relevant documents from the non-relevant ones. All these simplifications significantly reduce training times, but also improve ranking performance.

STB was trained and tested on MS MARCO. As the teacher model, BERT-Base was used, while two smaller models, L6-H768 and L3-H384, were used as students. Their document ranker is based on the work of Dai & Callan (2019), making use of their BERT-MaxP model. Below shows the performance of their models on the MS MARCO test queries.

Model	MRR@10	Size	Speedup
BERT-Base	0.3523	109M	1.0×
L6-H768	0.3848	67M	2×
L3-H384	0.3614	17M	15×

Table 3.2: Performance of Simplified TinyBERT on MS MARCO test queries.

Looking at the results, it is obvious that their method works, as both student models outperform the bigger teacher model. These results are analogous to the findings of Li et al. (2020), which applied KD to their PARADE model. Given these findings, knowledge distillation is most definitely useful for BERT-based document ranking models.

Chapter 4

Method

This chapter presents the research methodology that was used to answer the research questions. It aims to explain how all our experiments were conducted, demonstrating the reliability and validity of the results. This includes how BERT was fine-tuned, the setup we used to rank documents and how we evaluated the models. The rationale behind the choices that were made is also explained.

4.1 Fine-tuning BERT

The first thing that needed to be done, was to fine-tune the pre-trained BERT model. In order to do this, we had to choose a model architecture, create training data and find suitable hyperparameters.

4.1.1 Model Choice

The chosen model to use was a pointwise model, which was referred to as monoBERT in the previous chapter. It equips the pre-trained BERT model with a single output layer on top of the [CLS] token output, which after running the output through a softmax activation function, gives an output score between 0 and 1, which can directly be used as a relevance score.

To implement and train the model, the Huggingface transformers library was used. This library is easy to use and is well documented. A big benefit of using the pointwise model was that we could make use of the BertForSequenceClassification model, which is included in the library, including functionality for hyperparameter tuning and model training.

As input sequences, the query text was set as sentence A and a passage from a document as sentence B. The maximum input sequence length was set to 256, as this was faster than the allowed maximum of 512, but still let us use fairly long passages. The recommended 128 tokens would be too small for our purpose. 47 tokens were set aside for the query, and after adding the special tokens, [CLS] and [SEP], the remaining 207 tokens were left for the passage. Any space that was not used was filled by [PAD] tokens. We used the fast uncased tokenizer (known in the library as BertTokenizerFast), which contained 30522 tokens.

BERT comes in many sizes and configurations, and we chose to use BERT-Small. It is identical to BERT-Base, but it uses 4 transformer encodes, instead of 8 and the hidden size is reduced from 768 to 512. This reduced the number of parameters and therefore also training and inference times. Results from Li et al. (2020), when used in the PARADE model on the Robust04 dataset, showed that it was able to give satisfying results, which were only slightly worse than BERT-Base. This justified the choice of the smaller model.

Due to limited time and resources, we opted to use a relatively simple approach. This means that there exist other models that were not used in our experiments that would lead to better results than those reported in this thesis.

4.1.2 Training the Model

Training Data

To create training data, the MS MARCO document ranking dataset was used. Firstly, 400 labeled queries were randomly sampled from the 367013 available training queries. Afterward, the corresponding relevant documents were collected, together with 99600 randomly sampled documents, giving a total of 100000 documents. These documents were uploaded to a document ranking application, identical to the two first phases of the one that will be described in section 4.2.

Since BERT will be used to rank documents retrieved by BM25, the application was queried using the sampled training queries. The top 40 documents, as ranked by BM25, were used for each query. This number was chosen to mimic the number of candidate documents used in the BERT ranking application. Each document was divided into passages, inspired by the work of Dai & Callan (2019). If the document had a title that was shorter than 50 words (this constraint was added to ensure documents with very long titles were not divided into too many passages), it was placed in front of the passage. The total length of each passage was 150 words, with 50 words overlapping between passages. As a result of this process, 262787 query-passage pairs were collected. If the passage came from a relevant document, it would be labeled 1, and 0 otherwise.

Optimization and Hyperparameter Tuning

The model was trained to minimize cross-entropy loss, given by 3.2, and the default Adam optimizer with weight decay was used to do so. All the parameters of the model were adjusted. During training, 20 % of the samples were used to evaluate the model and the rest used by the optimizer. We used a GPU provided by the Idun cluster to perform the training, and fine-tuning BERT-Small took around 2.5 hours there.

To optimize the training process, hyperparameter tuning was performed. We used a grid search over the possible values, which were chosen based on the research conducted by Devlin et al. (2019) and Li et al. (2020). The chosen values were based on the model with the lowest evaluation loss. The results from the tuning are shown in table 4.1.

Parameter	Possible Values	Chosen Value
Learning Rate	1e-6, 5e-6, 1e-5, 5e-5	5e-6
Epochs	3, 4	3
Batch Size	16, 32, 64	16
Dropout	0.1	0.1
Weight Decay	0.01	0.01

Table 4.1: Results from doing hyperpameter tuning

4.1.3 Knowledge Distilled Models

To test the performance of knowledge distilled models, the Simplified TinyBERT models, as described in Chen et al. (2021), were used. The two distilled models and the teacher model (BERT-Base) were downloaded from the Simplified TinyBERT Github page, and used without any modifications.

The teacher model was trained on the MS MARCO passage ranking dataset, following the same setup as used in Nogueira & Cho (2019). The training data for the distilled models was created by splitting documents of the MS MARCO document ranking dataset into overlapping passages. For every relevant document, the five top-scoring query-passage pairs were labeled relevant. For every positive sample, a negative sample was created by randomly sampling passages from non-relevant documents. In total, 3.3M query-passage pairs were created.

The distilled models were fine-tuned for two epochs, using 64 as batch size and 5e-5 as learning rate. The rest of the hyperparameters and the optimization were identical to the one described in section 4.1.2. The maximum input sequence length was set to 256. We refer to the paper by Chen et al. (2021) for more details about how the models were trained.

4.2 Document Ranking Setup

The main goal of this thesis was to see if BERT could be used in a document ranking system. Therefore, the fine-tuned BERT model was used as the last phase of a bigger document ranking application that was used to generate the results in the experiments.

Our document ranking setup consisted of three stages. First, all candidate documents were retrieved using the **OR** method. Usually, this would retrieve too many documents, typically around 80 % of all documents. In our case, however, we wanted as high recall as possible, and this method minimized the risk of throwing away relevant documents.

The second phase was a BM25 ranking phase, based on equation 2.2 with $k = 1.2$ and $b = 0.75$. The number of documents returned from this phase is referred to as R, which was set to 20 in all experiments, except experiment 3b. The ranking from this phase served as a baseline for our BERT re-ranker to beat.

The last phase was the BERT model. The documents in the list returned by BM25 were first divided into passages, the same way as in the training data, and then fed into the model, one after the other. All the relevance scores for the passages of a document were aggregated, by using the mean or maximum passage score, to create a document relevance score. This score was then used to create the final ranked list.

The application was implemented in Vespa. Vespa is a customizable big data serving engine which we can use to build our own applications in. It is scalable, provides fast response times in addition to being easy to use when it comes to specifying our own application and the data used. We made use of pyvespa, Vespa's Python API, letting us use Python as programming language, which is preferred when performing ML experiments.

The implementation in Vespa was not as straightforward as first thought. Dividing up the documents and evaluating each one was not possible to do within the application, which complicated things. The solution to this problem was to create the passages locally and upload them as separate documents, using the same ID as the origin document. The passages were tokenized before they were uploaded, which meant time was saved when feeding them into BERT.

To query the app, document IDs were retrieved using BM25 on the full-size documents. The passages from the retrieved documents were then found by using these IDs. Finally, BERT was applied to each passage and the inference results were aggregated using Vespa's grouping functionality.

4.3 Experimental Procedure

4.3.1 Evaluation Data

The data we used to evaluate the models was collected the same way as the training data. 200 test queries were randomly sampled from MS MARCO, together with 200000 documents, including all the relevant documents. None of the queries or documents overlapped with the training data, such that all evaluation data was unseen to BERT.

All documents were uploaded to the document ranking application described above, and each document was divided into passages in the same manner as the training data.

4.3.2 Metrics

The primary metric we focused on was MRR@10, as each query only had one relevant document with a binary relevance judgment. In this case, MRR is equal to MAP. The number 10 was chosen since we only cared about the top-scoring documents. This metric indicated how good a model was at separating documents that got a high BM25 score, but were not relevant, from those that were actually relevant.

We also looked at the Recall@10 metric, which describes the proportion of queries in which the relevant document was placed among the top 10 ranked documents. This metric gave us an indication of how good the models were at separating the relevant documents from the not-so-relevant ones.

To test the statistical significance of our models, compared to the BM25 baseline, a two-tailed paired t-test was performed (Urbano et al. 2019). The number of test queries and the central limit theorem ensure that the assumption of normality is valid. The test was performed using `scipy.stats.ttest_rel`. All results that were statistically significant with $p < 0.05$ were marked with †.

To define the t-test, we let $B_1, \ldots, B_n$ be the baseline RR@10 values for each of our $n = 200$ queries with mean $\overline{B}$, and $E_1, \ldots, E_n$ be the RR@10 values for our experimental model with mean $\overline{E}$. We define $D_i = E_i - B_i$ and $\overline{D} = \overline{E} - \overline{B}$.

We assumed that both models perform equal and therefore set up the following null and alternative hypotheses:

$$H_0 : \mu_D = 0, \quad H_1 : \mu_D \neq 0$$

As test statistic, we used the T-statistic, which under H_0 is written as

$$T = \frac{\overline{D}}{\frac{S_D}{\sqrt{n}}} \sim t_{n-1}, \quad S_D = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(D_i - \overline{D}\right)^2}.$$

Chapter 5

Experiments and Results

In this chapter, we explain all the experiments that were conducted and their corresponding results. For each experiment, we also discuss the results by trying to explain the findings and how they relate to the theory. The results are also compared to results from similar experiments. In chapter 6, we summarize the findings of all the experiments and how they help answer the research questions.

Each of the three main experiments aims to answer the three research questions. Experiments 2 and 3 are divided into several smaller experiments that highlight different aspects of the research questions.

5.1 Experiment 1– Does it Work?

The first and most important question we tried to explore was – *did our setup work?* Were we able to fine-tune a BERT model that could replicate the findings of similar models? If our results are similar to related findings, we are able to be more confident in the results in the subsequent experiments.

In this first experiment, we compared the BM25 baseline setup against the two basic BERT models, BERT-Mean and BERT-Max, which refer to the way the score aggregation is done. The choice to use the mean instead of the sum of the passage scores was made to remove the unwanted effect that longer documents would get an unfair advantage.

Results from running the models using BERT-Small with $R = 20$ candidate documents are shown in table 5.1.

Model	MRR@10	Recall@10
BM25	0.517	0.710
BERT-Max	0.560	0.760
BERT-Mean	0.571†	0.765

Table 5.1: Results of models on 200 test queries in MS MARCO dataset. † denotes statistically significant results compared to BM25 ($p < 0.05$, two-tailed paired t-test)

To further showcase the differences between the models, the distributions of the ranks of the relevant documents were plotted as shown below.

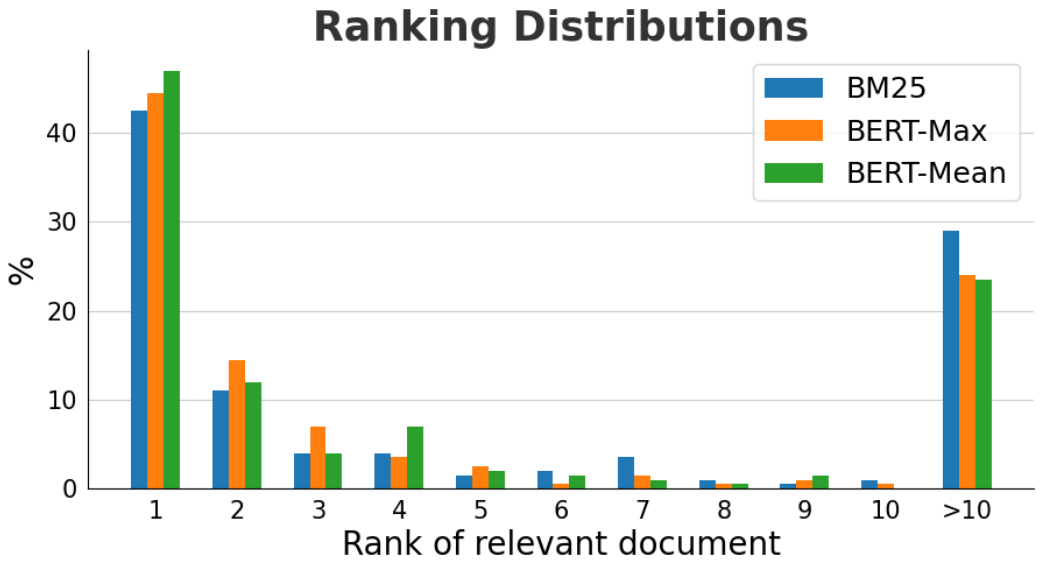


Figure 5.1: Ranking distributions of BM25, BERT-Max and BERT-Mean.

Firstly, we observe the increase in MRR the BERT models achieve, where the results of BERT-Mean are significant. This is also reflected in the recall metric, which explains some of the increase in MRR. The fact that the recall is higher, indicates that BERT is able to use its language understanding to find relevant documents that are not found by a term-matching method like BM25. For reference, the recall@20 score for BM25, which serves as the upper limit for how high recall BERT can achieve, was 0.79.

To try to further explain where the differences in MRR come from, except from looking at the recall, we can make use of the bar plot. As the RR metric is mostly affected by documents ranked close to 1, the rank 1, 2 and 3 sets of bars give us a great indicator of the MRR differences. BERT-Mean managed to get 94 queries perfect, compared to BM25's 85, and if we pretend that these 9 extra documents were not placed among the top 10 by BM25, it explains over 80% of the MRR gain. This is similar for BERT-Max, however, in this case, the rank 2 and 3 columns also contribute. The fact that BERT is better at placing relevant documents in the top ranks, tells us that it is able to filter out the relevant document from a set of documents with high BM25 scores, which was exactly what the model was trained to do.

We also see a difference between the two score aggregation techniques, albeit small and possibly random. However, comparing it to the findings of Dai & Callan (2019), which found that BERT-SumP worked slightly worse than BERT-MaxP, our results suggest that reducing the systematic bias towards longer documents by taking the mean is favorable.

As BERT-Mean was the best performing model in this experiment, all subsequent experiments are conducted using this aggregation technique.

5.2 Experiment 2 – Fine-tuning Procedure

The original BERT paper by Devlin et al. (2019) gave some information about how the model should be fine-tuned, but there were still a lot of things that were uncertain in the fine-tuning procedure for the document ranking task. The related models used for this purpose do not include details about how the fine-tuning was performed. We therefore found it suitable to run some experiments that aim to show how different factors in the fine-tuning procedure can affect the final ranking performance of a model.

A drawback of using a pointwise approach was that the metric we want to optimize during training, the cross-entropy loss, was not the same as the metric we use to evaluate the ranking performance. This meant that improvements in the CE-loss would not necessarily lead to improved MRR. As a consequence, we had to test all models on the evaluation application.

5.2.1 Experiment 2a – Which Parameters to Fine-Tune?

When we were fine-tuning BERT, we had the option of adjusting all the parameters in the model, or just the task-specific ones, i.e. the parameters in the added output layer. None of the models using BERT for document ranking give any information about this. Therefore, we tried out both alternatives, such that we know what works best when we are fine-tuning other models.

Fine-tuned Parameters	MRR@10
All BERT Parameters	0.571
Task-specific Parameters Only	0.520

Table 5.2: Results from fine-tuning all vs. task-specific parameters.

The results speak for themselves, which made it easy to choose to optimize all the parameters in all of our models. The number of task-specific parameters was likely too small to fit the training data well, and using the output of the [CLS] token without any modification was not sufficient to give good results. However, we see that it performs slightly better than BM25, which confirms the fact that BERT is able to understand natural language in its pre-trained state.

5.2.2 Experiment 2b – Overfitting

A typical issue when doing machine learning is *overfitting*. The problem arises when the model fits the training data very well, including the intrinsic noise in the data. Consequently, it is not able to capture the patterns in the data, and therefore performs poorly on unseen data. This can often happen with models with many parameters, as these are very flexible and can fit the data very well.

Given BERT's vast number of parameters, we therefore wanted to check if BERT for document ranking would suffer from overfitting. To test this, we trained a model for 6 epochs and compared it to the one trained for 3. To further inspect the fine-tuning, the training and evaluation losses were plotted for the two models.

Epochs	MRR@10	Final Eval Loss	Final Training Loss
3	0.571	0.0193	0.0186
6	0.542	0.0187	0.0067

Table 5.3: Ranking performance of models trained for 3 and 6 epochs.

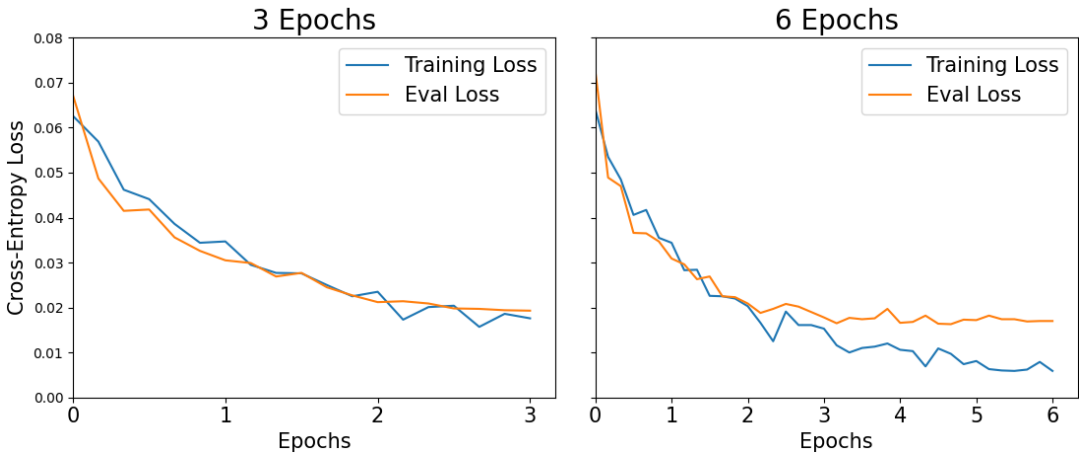


Figure 5.2: Training and evalution loss when training 3 and 6 epochs

The ranking results show that training the model for longer does not increase performance. Looking at the loss plots, the results are likely to be a consequence of overfitting. The loss functions in the model trained for 3 epochs behave like we want them to. The evaluation loss closely follows the training loss, showing us that the model performs equally well on the unseen data as on the training data. The evaluation loss is also very flat during the last epoch, indicating that the model has extracted as much knowledge from the training data as possible and that further training would not increase the model's performance.

However, after 3 epochs, the evaluation loss stagnates, while the training loss continues to decrease. This is a typical indicator of overfitting and helps explain the poor performance of this model. Consequently, we have to be careful with training our models for too long. These results justified the choice of 3 epochs in the hyperparameter tuning. They also suggest the use of early stopping, which means that we stop the training if we observe indications of overfitting.

5.2.3 Experiment 2c – Effect of Random Initialization

Even though the main BERT model has been pre-trained, the parameters in the output layer are being randomly initialized every time we train a model. This may lead to differences in performance, so we therefore tried training 4 identical models with different initial parameters, to see if there were any notable differences. We also plotted the evaluation losses for comparison.

Run	MRR@10	Final Eval Loss
1	0.571	0.0193
2	0.560	0.0186
3	0.564	0.0193
4	0.565	0.0182

Table 5.4: Ranking performance of 4 identical models with different initial weights.

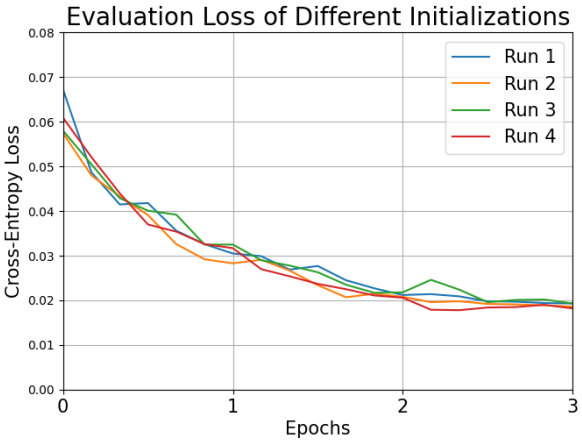


Figure 5.3: Evalutation loss of 4 identical models with different initial weights.

We see that there are differences in the ranking performance, which between the best and the worst are not insignificant. It is difficult to tell if this is a result of some models have a better semantic understand than others, or if they just happen to perform better on the test queries. Nevertheless, the results tell us that in order to train a good model, one should try to train several identical models and test which one works best. How many runs depends on the resources available. More test queries should also be considered to minimize variance in the MRR metric.

The evaluation losses are also interesting to look at. Firstly, we observe that the final loss value does not reflect how good the actual ranking performance is. This is especially evident between runs 1 and 2. Secondly, the plot also shows that the losses end up more or less equal. This is a good thing, telling us that the training is working as we want it, i.e. eliminating most of the effects the random initialization may bring. The fact that the values are not equal in the end is due to the randomness of the stochastic gradient descent optimization and is to be expected. Because the losses are so similar, the cross-entropy loss is not the best indicator of the ranking performance, and therefore the models should be compared using the actual document ranking system.

5.3 Experiment 3 – Speed vs. Performance

Although BERT improves overall ranking performance, it is slow to run, which is one of the major drawbacks of the model. Sometimes we want to trade off some of that additional performance to get faster inference times. In this section, we consider some factors that influence the speed, in order to see how it affects the performance.

5.3.1 Experiment 3a – Model Size

A reasonable first step would be to see how influential the size of the model is. For systems where limited memory might become an issue, smaller models are also preferred. BERT is available in a wide range of sizes and configurations, all sharing the same structure. The differences in the model sizes are affected by the number of transformer encoders (L) and the number of elements in the sequence vectors (H), which is also referred to as the hidden size. The number of attention heads (A) is chosen such that $H = 64A$.

In this experiment, we looked at the named models from the official BERT github page and one additional model. All the models had been identically pre-trained by Google and in this experiment, all models were fine-tuned equally as well, using the same training data and hyperparameters. We also measured how many queries per minute each model was able to evaluate, which included the retrieval phase and the BM25 ranking. However, the speed is predominantly affected by the BERT speed, as ranking with only BM25 is able to process around 100 queries per minute.

Model	L/H	MRR@10	Recall@10	#Parameters	Queries/min
BERT-Base	12/768	0.613†	0.770	109M	0.41
L4-H768	4/768	0.573†	0.760	53M	1.02
BERT-Medium	8/512	0.582†	0.760	42M	0.98
BERT-Small	4/512	0.571†	0.765	29M	2.01
BERT-Mini	4/256	0.551	0.775	12M	5.71
BERT-Tiny	2/128	0.538	0.770	4.4M	24.3

Table 5.5: Ranking results using various sizes of BERT. Statistically significant results compared to BM25 are marked with † ($p < 0.05$, two-tailed paired t-test).

The MRR results show a clear trend – a bigger model gives better performance. This is not surprising and the results are similar to the results when using PARADE with different BERT sizes (Li et al. 2020). The results are encouraging in the sense that even the tiny model, which is by far the fastest, is able to outperform BM25. It is also interesting to see that all the models perform equally when it comes to recall. This shows that the additional parameters are used to gain deeper knowledge that can make the model better separate the top-scoring documents.

The reason for adding the unnamed model (L4-H768), was to see how increasing the model size by adding more transformer encoders compared against increasing the hidden size. This comparison is based on the results from L4-H768 and BERT-Medium. We see that L4-H768 contains more parameters, but in terms of inference time, which is what we are most interested in, they compare very similarly. From the results, we see clearly that adding more transformer encoders makes a bigger impact than increasing the hidden size. This indicates that the additional encoders are able to extract more meaning, rather than just increasing the total parameter count by increasing the hidden size. This result is in contrast to the findings of Li et al. (2020), which found that the hidden size was more influential than the number of encoders. This discrepancy is likely due to the differences in the two model architectures, but it is difficult to tell without further exploration.

The performance results have to be observed in conjunction with the inference times. The computation times are probably not comparable to a high-quality retrieval system, but the relative differences should still be representative. Nevertheless, the last column clearly shows how much faster the smaller models are, which means that there is a lot of time to be saved, for a small drop in ranking performance.

5.3.2 Experiment 3b – Number of Documents to Rank

The time it takes to evaluate a query is also affected by how many documents BERT has to re-rank. The inference times increase linearly with the number of candidate documents, but how the performance is affected is not obvious. In this experiment, we tried to look at what happens with MRR@10 when the number of candidate documents ranges between 10 and 100. The results are shown below.

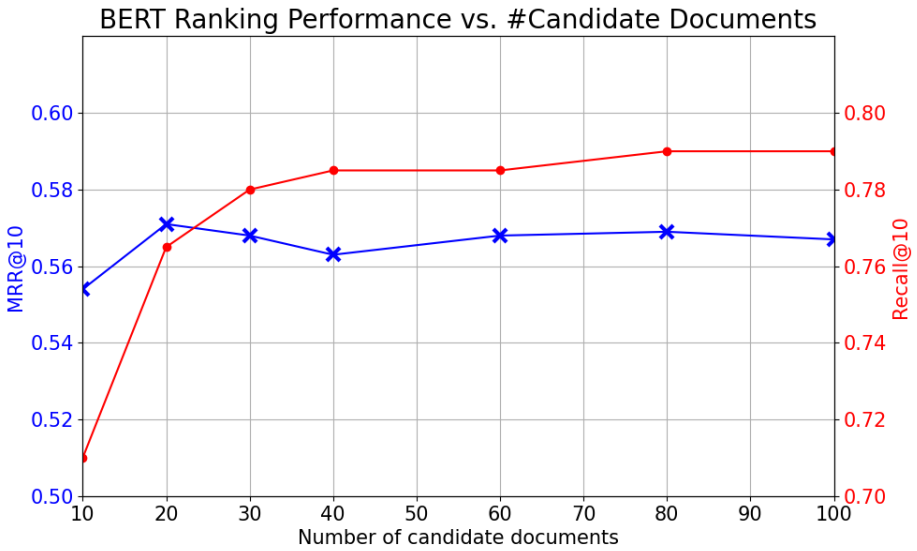


Figure 5.4: Ranking performance for varying number of candidate documents. The blue line shows MRR@10 (left y-axis) and the red is Recall@10 (right y-axis).

The figure shows the recall monotonically increasing with the number of candidate documents, which is a good sign. It tells us that our model is able to filter out the relevant document from a bigger pool of candidate documents and find more of the documents in the cases where BM25 falls short. With R equal to 80 and 100, the model is able to find all the relevant documents.

Interestingly, the improvement in recall does not lead to an increase in MRR, which slightly decreases and flattens out after 20 candidate documents. Even though the differences are very small, it could indicate that BERT becomes "confused" by these additional documents. A bigger BERT model, with a deeper understanding, could possibly overcome this unwanted effect, as it would better be able to separate these relevant documents from the confusing ones. Increasing the number of candidate documents per query when making the training data could also help.

In relation to speed, the results tell us that good performance is possible with a small number of candidate documents, which is good news. If we want to achieve better recall, we have to sacrifice some speed, but this tradeoff seems to give diminishing returns after 40 candidates.

5.3.3 Experiment 3c – Knowledge Distillation

Knowledge distillation techniques have been shown to enhance performance for BERT-based models within document ranking. We therefore wanted to test how well KD models perform in our ranking setup and check if our results are similar to other findings. We used Simplified TinyBERT's distilled models, L3-H384 and L6-H768, without any modifications. We also compared them against the teacher model, BERT-Base.

The results of Chen et al. (2021) did not include a direct comparison between how models fine-tuned with KD compare against the same models fine-tuned the regular way. We therefore fine-tuned the two student models as described in section 4.1.2. These models are denoted "Train" in the results. It is important to note that the distilled models have been fine-tuned longer on a bigger dataset, which could lead to increased performance. The differences in the way the training data was created, may also influence the results. Nevertheless, as the models are very similar and based on the same principles, we believe that fair conclusions can be drawn based on the results, which are shown in table 5.6.

Model	MRR@10(Train)	MRR@10(Distilled)	Size	Queries/min
Teacher	$0.619^{\dagger}$	-	109M	0.41
L6-H768	$0.585^{\dagger}$	$0.634^{\dagger}$	67M	0.81
L3-H384	0.564	$0.608^{\dagger}$	17M	5.73

Table 5.6: Performance of knowledge distilled models, including the teacher model (BERT-Base). The results from the models trained without knowledge distillation are marked "Train". Statistically significant results compared to BM25 are marked with † ($p < 0.05$, two-tailed paired t-test).

The results clearly show the usefulness of knowledge distillation, where both students achieve about 8% higher MRR scores than their regularly trained counterparts. It tells us that smaller BERT models are able to achieve impressive results, which makes them much more preferred to use in many retrieval systems. For future work, it would be interesting to see how much a small model like BERT-Tiny would benefit from KD.

The results also mirror the findings of Chen et al. (2021) in the sense that the L6-H768 clearly outperforming its teacher and the smaller model performing similarly as the teacher. Adding this to the fact that the students are 2 and 14 times faster, it is remarkable. Chen et al. (2021) do not give any explanation of why this happens, but we hypothesize that during the distillation, only the most important knowledge is passed down to the students, whereas the teacher contains unnecessary parameters that could be a source of error. The differences in the way the models were trained may also play a role.

Lastly, we observe that the teacher model performs very similarly to the BERT-Base we evaluated earlier (0.619 vs. 0.613). This shows that training the model using the MS MARCO passage dataset is very similar to training the model on the document dataset and then splitting the documents into passages the way we did.

Chapter 6

Conclusion

In this thesis, we have investigated how an attention-based model like BERT can use its language understanding to rank documents, which is a typical problem within information retrieval. Based on theory from transfer learning and innovations from the Transformer, BERT is able to improve on the two major drawbacks with traditional RNN-based sequence-to-sequence models, namely reducing training time by being parallelizable and removing short-term memory by using self-attention. By fine-tuning the model on data from the MS MARCO dataset, BERT is able to outperform classical document ranking methods.

In this final chapter, we discuss the most important findings from the experiments and relate them to the research questions and how they contribute to the knowledge in the field. We also discuss some improvement points in the research methodology. Finally, some thoughts and ideas for future work on the subject are presented.

6.1 Discussion

6.1.1 Evaluation of Research Questions

Research Question 1 – *How well do BERT-based document ranking models perform compared to classical, well-used ranking methods?*

The short answer to the question is: very well! BERT, with the help of attention mechanisms from the Transformer, is able to use its language understanding to separate relevant documents from the non-relevant ones, which was the main objective of this thesis.

In experiment 1, we showed that a pointwise ranking model using BERT-Small was able to significantly outperform BM25, improving MRR@10 by over 10% (0.517 vs. 0.571) and recall@10 with almost 8% (0.71 vs. 0.765). BM25 is a well-used baseline method within document ranking and the relative performance gains we achieved against this baseline are comparable with the results of Dai & Callan (2019), despite the fact that we used a smaller model.

We used passage score aggregation to create a relevance score for each document, and our results indicate that taking the average passage score works slightly better than using the maximum passage score. Other findings suggest that the maximum score outperforms the sum of the scores, which indicates that reducing the bias towards longer documents is favorable.

Research Question 2 – *What are important factors when fine-tuning BERT for document ranking?*

As a consequence of using the pointwise approach, adjusting all the parameters in BERT is necessary to achieve good performance. The number of task-specific parameters is too small to fit the data well.

We have to be careful not to overfit our models. BERT has a huge number of parameters and by training the model for too many epochs, we run into the risk of overfitting the model to the data. Comparing the results from training a model for 3 and 6 epochs, the models achieved MRR scores equal to 0.571 and 0.542, respectively, showing a clear difference in performance.

Lastly, we tried to explore the effects of randomly initializing the model weights. After training 4 identical models, there was a slight difference between the models. However, we suspect that this could also be a consequence of noise in the test data. Nevertheless, the final evaluation cross-entropy losses differ from the MRR performance, which means that we need to test all the models on the actual document ranking system to accurately measure a model's performance.

Research Question 3 – *How do factors that influence the speed of BERT impact the ranking performance?*

In experiment 3, we explored three different factors that influence the speed of BERT. The model size plays a major role in the performance, where the performance correlates well with the number of parameters. Not surprisingly, the speed decreases with the number of parameters. We also found that having more transformer encoder is more beneficial than increasing the hidden size, which contrasts the findings of Li et al. (2020). Differences in model architecture may explain this dissimilarity.

We also found that increasing the number of candidate documents does not give immediate improvements in ranking performance. The recall, however, did increase. The additional documents might have confused BERT, which means that a bigger model is needed to separate the relevant documents from the confusing ones. The differences were small, so further research on this is needed to make any hard conclusions.

The final experiment we conducted revolved around how knowledge distillation could be used in BERT-based ranking system, making use of the models trained by Simplified TinyBERT. With two smaller student models that were 2 and 14 times faster, the MRR scores were boosted to 0.634 and 0.608. Training the models the regular way, the models achieved MRR equal to 0.585 and 0.564, which was expected for models of their size. When comparing the biggest student model, L6-H768, to its teacher, BERT-base, the student performed better (0.634 vs. 0.619). This is in line with the findings of Chen et al. (2021). We hypothesize that using knowledge distillation only the most important is passed down to the student, whereas the teacher contains excess parameters that could be a source of error.

6.1.2 Improvement Points

Based on the findings from the experiments, we conclude that the chosen methodology, including the model choice, the creation of training data and the evaluation setup, was able to provide meaningful results that we could base our conclusions on. However, there were some improvement points that could contribute to more consistent and comparable results, as well as improving the model performances. The limitations discussed below were primarily due to time and computation constraints, but also as a result of knowledge gained throughout the implementation of the experiments.

We used a relatively small document corpus. The MS MARCO document ranking dataset contains 3.3M documents, and using just 200000 of them makes it much easier for the ranking model to perform well. This explains why our models achieve a much higher MRR than the models on the MS MARCO leaderboard. With more documents in the corpus, the number of candidate documents should also be higher than 20, as BM25 would struggle more to filter out the relevant ones. We hypothesize that BERT-based models would be better off when using more documents, as compared to BM25, but this is uncertain.

More test queries would give more significant results that were easier to base conclusions on. By using 200 queries, we were able to get statistically significant results, but some results were ambiguous. For example, in experiment 2c, it was difficult to conclude that the differences in performance were actually a result of different model initializations, or if they were just a consequence of variance in the test queries. However, the Robust04 test collection only contains 249 queries, which helps justify our choice.

Using more training data by increasing the number of training samples would probably increase performance. This is based on the great performance of the Simplified TinyBERT models, where the creators suggest that a lot of data is needed to train a model. More data would also reduce the risk of overfitting. Improving the way training data was created could also enhance performance. We based the creation of training data on the approach suggested by Dai & Callan (2019). In this method, all passages from a relevant document were considered relevant. However, not all passages from a relevant are in fact relevant, which means that we teach our models to find nonexistent relevance. This is in some cases mitigated by prepending the title, but not always. The approach used by Chen et al. (2021), tries to combat this issue by using a larger model to filter out the top five relevant passages, which are then used to train the other models. Using this technique, taking the maximum passage score would likely be a better aggregation technique than using the mean passage score.

6.1.3 Contributions

Despite the improvement points mentioned and the fact that we used a relatively simple ranking setup, the results in this thesis contribute to knowledge on the topic. Unlike other papers that use passage score aggregation, we considered taking the average passage score. We did this to remove the systematic bias towards longer documents we get by only taking the sum of the scores. Whereas previous research concluded that the maximum passage score works the best, our results landed at the opposite conclusion.

Research question 2 aimed to fill in gaps in the research, as previous research provides very little information about the fine-tuning procedure itself. We believe that all the findings from experiment 2 are relevant for future research on the topic. The knowledge is also applicable to research that aims to fine-tune BERT for other tasks outside of document ranking and information retrieval.

Lastly, in our experiments, we used BERT-Small, which we found to perform worse than the bigger models. All other research on the topic so far has used either BERT-Large or BERT-Base, which are able to achieve better and more significant results. However, in this thesis, we showed that a smaller model is capable of providing useful knowledge in this field. As a consequence, we show that you do not need the biggest BERT model to be able to conduct research or improve a document ranking system. Ultimately, this makes it more applicable for systems where fast inference times are important or systems with limited computational resources.

6.2 Future Work

Considering the short time BERT has existed, the progress that has been made on the model is quite astounding. Encouraged by the impressive results BERT achieved within document ranking, a lot of researchers devoted their interest in the model, which has pushed the boundaries for the state of the art in the field. However, there are still a lot of areas to explore and questions to answer and it will be very interesting to see where the research will lead. We therefore present some thoughts for future work on the topic.

Explore different retrieval methods that work better with BERT. Most models use BM25 to retrieve a list of candidate documents, but this might not be the best method to use together with BERT. One alternative is to use approximate nearest neighbor search, which is a topic we did not explore in this thesis. In that approach, we use BERT to create a dense vector embedding of the document and the query and use a similarity metric, for example, the cosine similarity (2.1), to find candidate documents. By creating the document embeddings beforehand, and with the help of approximate search algorithms, the retrieval is very fast. Research has been done on this method, but more progress needs to be done.

There could be more sophisticated ways to use documents as input. All the models we looked at split documents into passages and use these to infer knowledge about the document. However, by using a fixed passage length, we risk splitting a sentence in half and thus lose meaning, although this is mitigated by using overlapping passages. A more natural way to split documents is by making use of sections or paragraphs, which most documents are made up from. Nevertheless, it is not unlikely that there exists a completely different way of solving this problem that has not been explored yet.

In order to use BERT in a general-purpose text search application, it has to be available for more than English, which the current research is predominantly based on. Pre-trained BERT models are available in a few other languages, for example, German and Chinese, which makes it possible to train language-specific models. However, the resources for document ranking, both test collections and training data, are very limited in most languages except English, and creating new data is time consuming. Consequently, this suggests the creation of cross-lingual ranking models, which are able to work on most languages. This could for example be done by using robust machine translation models.

BERT clearly shows how attention mechanisms can be used to understand relevance between query and document. This begs the question of whether creating an attention-based model specifically for this purpose is better than fine-tuning BERT. By sticking to BERT, researchers are able to take advantage of the advancements in the knowledge of BERT and allow for the use of transfer learning, i.e. only needing to fine-tune the model. The other approach allows for smaller and faster models because all unnecessary components are removed. Other attention-based models may also arise in the future that work better than BERT.

Bibliography

Ba, J. L., Kiros, J. R. & Hinton, G. E. (2016), 'Layer normalization'.
URL: https://arxiv.org/pdf/1607.06450.pdf *[Accessed 19.02.2021]*

Bahdanau, D., Cho, K. & Bengio, Y. (2015), 'Neural machine translation by jointly learning to align and translate'.
URL: https://arxiv.org/pdf/1409.0473.pdf *[Accessed 10.02.2021]*

Bajaj, P. et al. (2018), 'Ms marco: A human generated machine reading comprehension dataset'.
URL: https://arxiv.org/pdf/1611.09268.pdf *[Accessed 23.03.2021]*

Chen, X., He, B., Hui, K., Sun, L. & Sun, Y. (2021), Simplified tinybert: Knowledge distillation for document retrieval, *in* 'ECIR (2)', Vol. 12657 of *Lecture Notes in Computer Science*, Springer, pp. 241–248.
URL: https://arxiv.org/pdf/2009.07531v2.pdf *[Accessed 25.05.2021]*

Cho, K. et al. (2014), 'Learning phrase representations using rnn encoder–decoder for statistical machine translation'.
URL: http://emnlp2014.org/papers/pdf/EMNLP2014179.pdf *[Accessed 09.02.2021]*

Dai, Z. & Callan, J. (2019), 'Deeper text understanding for ir with contextual neural language modeling'.
URL: https://arxiv.org/pdf/1905.09217.pdf *[Accessed 08.03.2021]*

Datta, J. (2010), 'Ranking in information retrieval'.
URL: https://www.cse.iitb.ac.in/archive/internal/techreports/reports/TR-CSE-2010-31.pdf *[Accessed 12.04.2021]*

Devlin, J. et al. (2019), 'Bert: Pre-training of deep bidirectional transformers for language understanding'.
URL: https://arxiv.org/pdf/1810.04805.pdf *[Accessed 19.02.2021]*

Eisenstein, J. (2018), *Natural Language Processing*, MIT Press.
URL: https://cseweb.ucsd.edu/~nnakashole/teaching/eisenstein-nov18.pdf *[Accessed 02.02.2021]*

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press.
URL: http://www.deeplearningbook.org *[Accessed 03.02.2021]*

Jiao, X. et al. (2019), 'Tinybert: Distilling bert for natural language understanding'.
URL: https://arxiv.org/pdf/1909.10351.pdf *[Accessed 25.05.2021]*

Lan, Z. et al. (2019), 'Albert: A lite bert for self-supervised learning of language representations'.
URL: https://arxiv.org/pdf/1909.11942.pdf *[Accessed 14.05.2021]*

Li, C. et al. (2020), 'Parade: Passage representation aggregation for document reranking'.
URL: https://arxiv.org/pdf/2008.09093.pdf *[Accessed 09.03.2021]*

Lin, J., Nogueira, R. & Yates, A. (2020), 'Pretrained transformers for text ranking: Bert and beyond'.
URL: https://arxiv.org/pdf/2010.06467.pdf *[Accessed 04.03.2021]*

Liu, Y. et al. (2019), 'Roberta: A robustly optimized bert pretraining approach'.
URL: https://arxiv.org/pdf/1907.11692.pdf *[Accessed 14.05.2021]*

Manning, C. D., Raghavan, P. & Schütze, H. (2009), *An Introduction to Information Retrieval*, Cambridge University Press.
URL: https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf *[Accessed 04.02.2021]*

Mikolov, T. et al. (2013), 'Efficient estimation of word representations in vector space'.
URL: https://arxiv.org/pdf/1301.3781.pdf *[Accessed 03.02.2021]*

Nogueira, R. & Cho, K. (2019), 'Passage re-ranking with bert'.
URL: https://arxiv.org/pdf/1901.04085.pdf *[Accessed 04.03.2021]*

Nogueira, R. et al. (2019), 'Multi-stage document ranking with bert'.
URL: https://arxiv.org/pdf/1910.14424.pdf *[Accessed 08.03.2021]*

Robertson, D. & Zaragoza, H. (2009), *The Probabilistic Relevance Framework: BM25 and Beyond*, now publishers.

Sanh, V. et al. (2019), 'Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter'.
URL: https://arxiv.org/pdf/1910.01108.pdf *[Accessed 14.05.2021]*

Själander, M., Jahre, M., Tufte, G. & Reissmann, N. (2019), 'EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure'.

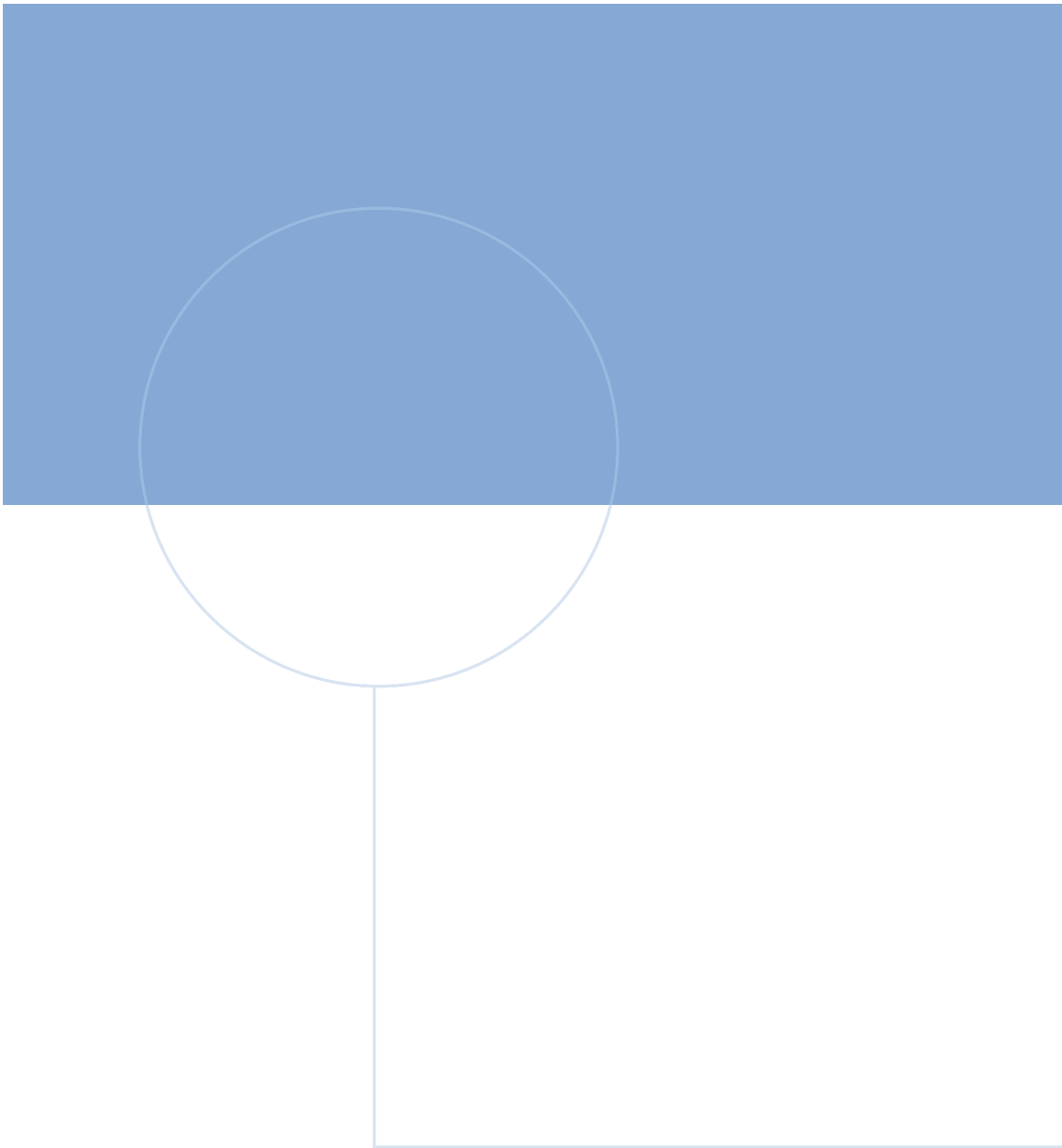
Sutskever, I., Vinyals, O. & Le, Q. V. (2014), 'Sequence to sequence learning with neural networks'.
URL: https://papers.nips.cc/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf *[Accessed 09.02.2021]*

Urbano, J., Lima, H. & Hanjalic, A. (2019), 'Statistical significance testing in information retrieval: An empirical analysis of type i, type ii and type iii errors'.
URL: https://julian-urbano.info/files/publications/076-statistical-significance-testing-information-retrieval-empirical-analysis-type-i-type-ii-type-iii-errors.pdf *[Accessed 11.05.2021]*

Vaswani, A. et al. (2017), 'Attention is all you need'.
URL: https://arxiv.org/pdf/1706.03762.pdf *[Accessed 12.02.2021]*

Wolf, T. et al. (2020), 'Transformers: State-of-the-art natural language processing'.
URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.pdf *[Accessed 12.02.2021]*

Wu, Y. et al. (2016), 'Google's neural machine translation system: Bridging the gap between human and machine translation'.
URL: https://arxiv.org/pdf/1609.08144.pdf *[Accessed 04.02.2021]*



NTNU

Norwegian University of
Science and Technology