Radboud University Nijmegen

Faculty of Science

The BERT Ranking Paradigm: Training Strategies Evaluated

MSc Thesis

Author:
M.J.P. Verbrugge
1044326

Supervisors:
Prof. dr. ir. D. Hiemstra
Ir. N. Ghasemi

Second reader:
Dr. ir. F. Hasibi

A thesis submitted in fulfillment of the requirements for the degree of Master of Science in Computing Science

November 15, 2021

Abstract

This thesis researches the most recent paradigm in information retrieval, which applies the neural language representation model BERT to rank relevant passages out of a corpus. The research focuses on a re-ranker scheme that uses BM25 to pre-rank the corpus followed by BERT-based ranking, exploring better fine-tuning methodology for a pre-trained BERT. This goal is pursued in two parts, in the first, all methods rely on binary relevance labels, while the second part applies methods that rely on multiple relevance labels instead. Part one researches methods that apply training data enhancement and the application of inductive transfer learning methods. Part two researches the application of *single class multi label* methods, *multi class multi label* methods and label-based regression. In all parts, the methods were evaluated on the fully annotated Cranfield dataset.

This thesis demonstrates that applying inductive transfer learning with the Next Sentence Prediction task improves the baseline by presenting various methods to enrich the fine-tuning data for different levels of the BM25-BERT ranking pipeline. Also, this thesis demonstrates that application of a regression method results in above baseline performance. This indicates the superiority of this method over rule-based filtering of classifier results.

Contents

Chapter 1

Introduction

1.1 Problem

This thesis focuses on recent developments in ranking for Information Retrieval: new methods that make use of modern neural networks for natural language processing (NLP) such as BERT [4] show impressive results. However, being tested on only partial annotated results, the proposed networks may achieve scores that are unfair and biased [23, 5]. For this reason, this thesis uses the fully annotated Cranfield collection.

The problem is that there is little knowledge of the unbiased performance of BERT used for ranking purposes; in other words, there is little knowledge about what the best strategy is to train a BERT-based ranker. Most experiments use data sets that are annotated using the *pool method*, which, given the construction date, did not take the neural rankers into account[23]. The focus of this thesis will be BERT fine-tuning for the ranking task with a fully annotated set, the Cranfield collection. Furthermore, another important subject will be to investigate whether the multiple relevance levels in the Cranfield collection can be leveraged during training.

The next paragraph will give some background on ranking and will formulate research questions. Besides this thesis, all the used code, with more explanations in the form of notebooks along with all the runs and their output can be found in the repository [1].

1.2 Background and Motivation

Searching and finding relevant (passages in) documents given a certain query is probably a problem that has been investigated since there are collections, such as libraries. Since the 1940s, there have been efforts that used 'electromechanical and computational devices that search manually generated catalogs' [15]. With the World Wide Web introduction, search engines were developed to help users find their way in the explosive growing amount of content. Before the neural network paradigm, the search algorithms were built on classic ranking mechanisms [3].

Nowadays, one of the new paradigms is the application of language representation models for search. An algorithm of particular interest is called BERT [4]. Researchers were able to achieve state-of-the-art performance using this model; a paper of particular interest for this research is 'Re-ranking with BERT' by Nogueira and Cho (2019) [8]. Although the model is novel, commonly used data sets such as Robust04 [20] date back to a pre-neural network 'era,' at least for IR[23]. Training a neural ranker with neural networks requires at least weakly annotated data sets [3] or better: fully annotated. This

[1] `https://github.com/MauiGonzo/BERT-BM25-Thesis-Project`

thesis builds on the neural ranker setup Nogueira and Cho (2019) presented, adapted by Ghasemi and Hiemstra (2021), to be trained and tested with the fully annotated Cranfield collection. The referenced works researches 1) a re-ranker set up that first makes a BM25@N search result which is then re-ranked by BERT and 2) a full ranker. This thesis focuses on the re-ranker setup or *pipeline* as it is called, which will be discussed in more detail in section 1.3. The benefit of re-ranking is that BERT only has to rank N documents per question instead of the whole corpus. This benefit is not only limited to training but also inference. At the same time, this can be seen as a drawback, i.e., possibly BM25@N misses a relevant document. This way, several documents are not seen during training, therefore compared to full ranking, re-ranking is not able to reach the same performance. Possibly it might be beneficial to show BERT all the relevant documents during the training. The re-ranking pipeline can be modified to address this issue, such that BM25-results can be enriched with relevant documents. This leads to the first research question: **RQ 1.1** - *Would it be beneficial to enrich the BM25-results with 'missed' relevant documents?*

Besides altering the training set, this thesis seeks to fine-tune BERT in other ways. It tends to leverage the ability to train BERT for different tasks: **RQ1.2, RQ1.3** *Would it be beneficial to, first, fine-tune BERT on another task, such as Masked Language Modeling (MLM) or the Next Sentence Prediction (NSP) - also known as inductive transfer learning?* These will be the second and third research questions, pointing to either MLM and NSP. The idea is that with the given small training set it may be beneficial to apply transfer learning by training the model on a closely related task [22, 10, 1].

The following paragraph presents a set of research questions based on the fact that the Cranfield collection has multiple relevance labels, i.e., for each relevant document is has a relevance score, more details will be discussed in Section 1.4. Instead of training with binary targets, as presented in [5], this work investigates whether it is beneficial to train BERT with multi-label targets. The general research question **RQ 2** *Would it be beneficial to use the multi-label relevance scores in the Cranfield collection for the BERT re-ranker?*. This is being researched in various manners; a distinction can be made in the type of model:

- **RQ 2.1** - *Would it be beneficial to train BERT with single class, multi-label targets?*
- **RQ 2.2** - *Would it be beneficial to train BERT with multi-class, multi-label targets?*
- **RQ 2.3** - *Would it be beneficial to train BERT on the relevance level directly, doing regression?*

1.3 BM25-BERT ranking pipeline

This section introduces the *re-ranking* pipeline that is used throughout this thesis. The following paragraph introduces the building blocks of the pipeline, which is based on [5]. The description here is the baseline set-up; the experiments will alter the baseline and describe the effect. Therefore, where needed, more details will be presented in later sections.

1.3.1 Re-Ranker Building Blocks

This section will incrementally introduce the re-ranker. First, the *full* ranker is introduced, which can be seen as a simpler version of the re-ranker. The re-ranking architecture is based on a scheme introduced by [8, 5], which also introduces the full-ranker.

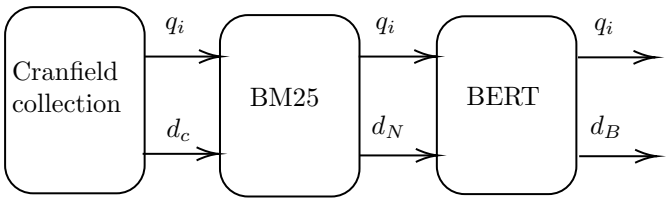

Figure 1.1: Schematic and simplified overview of re-ranker. Starting with the Cranfield collection, the scheme is read as follows: for every query q_i, any document in the corpus d_c is evaluated by BM25@N, which in turn outputs the best matching N documents d_N. The last step is a re-ranking with BERT, during which the set d_N is re-ranked by the model resulting in the ranking d_B. For the ease of reading the tokenization steps are left out.

The basic approach to ranking with BERT is the full-ranker setup, where the input is a query-document pair, and the output is a score that tells how well those relate. The sentences are tokenized before passed to BERT. More details on tokenization are given in Section 1.6. The output of BERT is a 768 long vector[4], to get anything useful out of BERT an extra layer is added that (simply put) reduces this vector to a score, also known as *logit*. This layer is called the *task head* and 'puts BERT to the task', in this case deriving relevance scores. Section 1.6 gives a further introduction on BERT.

This paragraph will describe the re-ranker, starting with a motivation to prefer this scheme, namely the drawback of the full ranker is that, for each query, all documents have to be evaluated; this requires substantial resources. Therefore a pre-selection step will be introduced: a BM25-ranker, which has the task to reduce the number of documents to N. A schematic representation of the re-ranker is given in Figure 1.1.

1.3.2 Ranking Process

The previous section presented the building blocks for the re-ranker; this section explains how the re-ranker calculates a performance figure. The data is split into five folds to get the most out of the relatively small Cranfield collection. For the remainder of this paragraph, first, the training is discussed, followed by the testing phase. During training, the BM25 receives all queries assigned to a fold, ranks all documents in the corpus, and forwards the best N to BERT. Henceforth, BERT trains, supervised, on this set of query-documents pairs. The testing phase is organized in a similar fashion. BERT has an input of N documents per query; using an equal batch size, they are sorted, after which the M best scoring documents are fed in an evaluation metric that returns a ranking score. The scores of all queries for a particular fold are combined, and finally, an average over all folds is given: the final performance for a particular configuration of the pipeline.

The setting for the folds, as well as the train and test settings (i.e., the random seed) are similar to those used in [5], making them easily comparable. More details on this subject will be discussed in Section 1.8.

1.4 Cranfield Dataset

The Cranfield collection[2] used in this thesis contains a fully annotated set of queries and documents of the College of Aeronautics, Cranfield, England [11]. This is the same set as used in [5]. The collection is comprised of the following:

[2] `http://ir.dcs.gla.ac.uk/resources/test_collections/cran/`

- a corpus of 1400 abstracts of documents with a aeronautical subject;
- a list of 256 queries, also on aeronautical subjects;
- a relevance table that (among others) indicates the annotated relevance of document d to query q.

The documents in the corpus are consist of a title sentence followed by the abstract of the article. Information Retrieval theory however uses the terms *document* and *corpus*, therefore this thesis will use those terms. The documents relevance codes defined by Cleverdon[3] (designer of the set) are given by Table 1.1. This overview indicates the relevance levels. When a document is not assigned a code, implicitly it is deemed not relevant. The code used during the thesis experiments assigns a code 0.

code	explanation
1	References which are a complete answer to the question.
2	References of a high degree of relevance, the lack of which either would have made the research impracticable or would have resulted in a considerable amount of extra work.
3	References which were useful, either as general background to the work or as suggesting methods of tackling certain aspects of the work.
4	References of minimum interest, for example, those that have been included from an historical viewpoint.

Table 1.1: The codes used for describing relevance of the Cranfield documents. One code, 0, is a bit special, it indicates not relevant, the dataset assign this code implicitly.

1.5 BM25

This Section gives a basic description of the BM25 algorithm, which goal is returning a ranked set of relevant documents given a query and a corpus. The BM25 ranking algorithm was developed in the 1970s and 1980s by Stephen E. Robertson, Karen Spärk Jones et al[4]. This Best Matching model ranks a requested number of documents in a corpus[5] given a query, i.e., it will give you the best matching N documents of the whole corpus, given a question. To do so, BM25 first transforms every document in a *bag-of-words* that is an array of the unique words found and an equal length array containing the frequency of occurrence in that document. A scoring function of document D and query Q, which consists of keywords $q_1...q_n$ is applied to every document. Based on the frequency of keywords and specific free parameters, every document is assigned a score. Then the top-ranking N documents are selected as a result. This section follows the symbols used on the referenced Wikipedia page. The formula below gives the BM25 formula:

$$\text{score}(D,Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D) + k_i \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

where the term frequency in document D is denoted by $f(q_i, D)$, the document length is $|D|$, avgdl is average document length of all documents in the corpus. There

[3] `https://en.wikipedia.org/wiki/Cranfield_experiments` (last visit: 5-11-2021)

[4] `https://en.wikipedia.org/wiki/Okapi_BM25` (last visit October 8, 2021) [12]

[5] The term corpus is generally used in Information Retrieval and indicates a set of documents, paragraphs, or (readable parts of) websites. Usually, every item of this set is called *document*

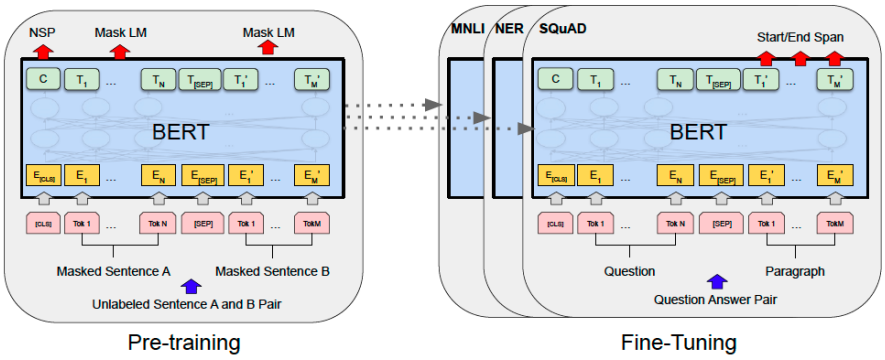

Figure 1.2: The illustration used in the original paper (Devlin, 2019). In other words, the scheme presents the procedures for pre-training and training, which use the same architecture. Going from bottom to top: a a-b pair is tokenized and a special starting-, separator- closing token. This input is passed to the encoder which processes it bi-directional every layer.

are two free parameters: k_1 and b. The abbreviation IDF indicates Inverse Document Frequency:

$$\text{IDF}(q_i) = ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

Where $n(q_i)$ indicates how many documents contain this keyword. Although this rule-based ranker is not very modern, it is light-weight, easily tuneable and performs well given the resources required. For this reasons it is still popular.

1.6 BERT

This section gives a high-level description of the BERT language representation model[4]; its history, the building blocks, and the tasks. This work limits its description of BERT to a point necessary to understand what methods this research used for the experiments. When Devlin et al published their paper, they also released the source code of BERT, alongside with pre-trained models. A short description of the algorithm is given before describing the benefit and applications of BERT, a schematic overvieuw is given by Figure 1.2.

First of all, the algorithm builds on the shoulders of giants; it is beyond the scope of this thesis to describe all the details; therefore, following the paper, this paragraph omits an extensive elaboration. The core of the algorithm is a bi-directional multilayer Transformer Encoder (it assess text from both left to right direction and vice versa), which in turn is a stack of a number[6] of attention layers. This is a complicated method that encodes input text into a representation that enables the model to understand the text given. Namely, a model cannot calculate document scores using words directly. However, it can use the encoded representation and hence perform natural language tasks. More detailed descriptions of the model can be found in the original implementation by Vaswani et al (2017)[19] and publications such as "The Annotated Transformer," "The Illustrated BERT, ELMo, and co." by Jay Allamar and for example the "Bert Research" videos and blogs by Chris McCormick.

[6]The paper has a main focus on Bert-Large which has 16 attention layers, and Bert-Base with 12 layers.

BERT Tokenizer

The basis of the model is the stack of encoders. By adding a top layer (the task head), BERT can be 'trained downstream' to perform various tasks, such as ranking, Masked Language Modelling (MLM), or Next Sentence Prediction (NSP). The input consists of (pieces) of text which are then tokenized by the BertTokenizer (that in turn uses WordPiece embeddings with a vocabulary of 30,000 tokens[21]). During training, the input has to remain constant; a variable `max_length` in the tokenizer controls the maximum length of the input. In case the input is shorter, the input will be padded with zero's.

Earlier in the introduction of the re-ranker, the query-document pairs, the input of BERT, were discussed. When combining this with the max length setting is may become clear that the combined pieces of text potentially contain more text or tokens than fit in the max length. This is dealt with by truncating the document because the question is important and considered short most of the time.

Training BERT

Contrary to most machine learning models, that come with empty parameters, such as U-Net[13] for image classification, the BERT-model is *pre-trained*. When any BERT model is instantiated, it will download the weights automatically. A specific version of the model can be chosen depending on the task. This means that a specific layer is added on top of the encoder, the *task head*. While the encoder is pre-trained, this task-specific layer is randomly initialized, so it has to be trained; this is called *fine tuning*. Usually, this is done for 1 to 3 epochs and will take around 1 hour, depending on the GPU type.

Why is the model pre-trained by default? Although it is possible to pre-train BERT yourself, pre-training is not feasible nor required for most researchers: BERT was pre-trained on the BooksCorpus[25] (800M words) and the English Wikipedia (2,500M words). Training of each BERT version takes four days and requires 4 to 64 TPUs (depending on the model's size). The cost of training BERT-Large is around $30k[17, 16].

1.7 Evaluation Metrics

The following paragraphs present a brief description of the metric used in this thesis: Normalised Discounted Cumulative Gain (NDCG) [7, 2].

The NDCG is based on the following two assumptions (as described by Croft, 2010):

- Highly relevant documents are more useful than marginally relevant documents.
- The lower the ranked position of a relevant document (i.e., further down the ranked list), the less useful it is for the user, since it is less likely to be examined.

This implies that when multiple documents have the same ranking with categorical relevance levels, the order between them does not matter according to this algorithm. The following expressions show how the NDCG is calculated, the first step is the Discounted Cumulative Gain (DCG):

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2 i}$$

Where p denotes the rank, one can think of the top result in a search ranking as $p = 1$ and next best as $p = 2$ and so on. The variable rel_i indicates the relevance level of the document retrieved at rank i, the Cranfield collection has a code system that goes from "Complete answer" to "... minimum interest" which is coded in four steps from 1

to 4. What is left is coded 0 by default and indicates 'not relevant'. Note that a good working of the formula above requires an ascending continuous relevance scale.

The normalized DCG is defined as:

$$\text{NDCG}_p = \frac{\text{NDCG}_p}{\text{IDCG}_p}$$

Where the IDCG is the ideal DCG, which can be written as:

$$IDCG_p = rel_1 + \sum_{i=2}^{|p|} \frac{rel_i}{log_2 i}$$

Which is very similar to the DCG_p but now the ordering is the perfect order i.e. all the highest relevance scores are put first. Where the DCG_p score derived over e.g. the first $p = 5$ ranked documents, the IDCG has to take into account all the documents available to derive the perfect ranking.

Finally a remark on the notation: in this thesis and IR theory the notation NDCG@N is usually used to indicate the NDCG-performance up to and including rank $p = N$.

1.8 Experiment Overview

In general, this thesis is a composition of methods that aim at improving the overall performance of the BM25-BERT ranking pipeline. Various sections of the pipeline are changed one at a time to measure and compare the impact of every method, this will be the content of Chapter 2 and 3.

parameter	*settings*
model-type	`bert-base-uncased`
learning rate	[2e−5, 3e−5]
epochs	[1, 2]
max length	128
batch size	32

Table 1.2: Settings used to fine tune BERT in the BM25-BERT pipeline setting. With two values for learning rate and epochs there are four combinations of hyperparameter settings.

1.8.1 General settings

During each experiment, two hyper-parameters are varied; this leaves four combinations. Table 1.2 presents an overview of those settings. All the (final) experiments were conducted on a Virtual Machine on Google Cloud Platform (GCP); this was a 2 CPU, 7.5Gb RAM machine with an NVIDIA TESLA T4. The repository contains the used notebooks containing all the detailed results.

1.8.2 Experiments

Chapter 2 has a focus on the input side of the re-ranker pipeline: namely the training set, BERT is typically trained with the BM25@N results, the candidate documents. Here, the idea is to enrich this list of candidate documents, labeled as relevant but missed by BM25@N. Following this approach, another method is used to enrich the model using the available documents, namely applying *inductive transfer learning* by training

BERT on different tasks first, being Masked Language Modeling task (MLM) and Next Sentence Prediction (NSP). The latter approaches transfer learning with various data sets, among others leveraging titles as special sentences that relate positively to all document sentences. Since MLM and NSP require different task heads, only the encoder weights are copied; the technical details of this are given in the code (notebook).

Chapter 3 presents methods that seek to leverage the multiple relevance levels that the Cranfield data set provides; these methods research beneficial use of the labels, starting with the default task head for ranking, up to custom task heads that feature various loss types and *multi tasks* approaches. Incorporated in this multi-label approach is the 'post-processing' of the predictions $\hat{y}$ that seek possibilities beyond the commonly used $arg\,max$ approach.

Contrary to *classifying* documents, the *regression* section explains a method that translates the classification problem into a regression problem. It leverages the multiple labels during training, using a conversion table, and returns one list of documents graded for relevance.

1.9 Organisation

This section describes the organization of this document. The introduction already presented the various building blocks of the BM25-BERT based re-ranking pipeline. It does not aim at an extensive discussion but serves to support the methods used in the Chapters hereafter. Following the introduction, Chapter 2 researches how to fine-tune the re-ranker by first 'enriching' input data; this method aims at adding relevant documents missed by the BM25 pre-ranking step. Secondly, it investigates the added value of training the model with the same data but different tasks.

Chapter 3 researches the incorporation of the multi-relevance labels. The Cranfield collection indicates with multiple relevance levels the relation between query and document. This chapter presents various methods to use all the relevance levels; so far, the baseline setup only used a binary transformation of the levels: indicating that a given document is relevant or not [5, 8]. The methods presented include multi-label, multi-class - multi-label, and regression.

The last chapter concludes the thesis by answering the research questions and discussing leads for future work.

Chapter 2

Enrich Training Sets and Transfer Learning

2.1 Introduction

This chapter investigates first the usefulness of having more data available at the input of the re-ranker pipeline. Therefore, it 1) explores whether it may be beneficial to adjust the training set using the relevant documents BM25 failed to find and 2) train the model with the same data on a different task before training it on ranking. Both methods rely on binary relevance labels, methods that rely on multiple relevance labels will be the subject of Chapter 3.

BERT comes with a pre-trained encoder, though it has to be trained downstream for a specific task. This is called transfer learning: "*a machine learning method where a model developed for a task is reused as the starting point for a model on a second task*"[1]. The following methods describe a variant on this: *inductive transfer learning* using the same domain. In general, training BERT for a task takes the following steps: first, we take a pre-trained BERT-encoder; one can download such a model, which Google already trains. This is common practice because it takes 24 hours on 16 Tensor Processing Units to pre-train this model ($\text{BERT}_{\text{BASE}}$) [4]. Secondly, we train this model for a natural language task, and thirdly we train it further for the ranking task. Section 2.4 describes how this is done with the Masked Language Modeling task (MLM), and the section after that will do the same for Next Sentence Prediction (NSP)—followed by a general section that compares the results.

2.2 Enrich BM25-results

Incorporating Missed Documents

The BM25-BERT re-ranker constructs for every fold a training set that is based on the documents retrieved by BM25. When comparing the retrieved documents with the ground truth, it turns out BM25 does not always retrieve all relevant documents, those we call *missed documents*. In other words, for each query, the list of documents BM25@100 produces does not include all documents labeled as relevant. The following describes two methods to incorporate those missed BM25-results.

The first research question about adding missed relevant documents to the BM25 results is covered in this section.

[1] https://machinelearningmastery.com/transfer-learning-for-deep-learning/ (last visit: 23-9-2021)

learning rate	2e−5		3e−5	
epochs	1	2	1	2
BM25@100	0.5445	0.5482	0.5439	**0.5486**
BM25@100-swap	0.5332	0.5307	0.5360	0.5353
BM25@100-add	0.5394	0.5246	0.5192	0.5296

Table 2.1: NCDG@20 test results after fine tuning with training sets that contain relevant documents missed by BM25, except for the first row, which can be seen as the baseline. The test set is always the same BM25@100 set (also used in [5]). The best NDCG score is formatted in bold. Other settings during the experiment were: `max_length`=128, batch-size=32. Furthermore, it took a TESLA T4, 38 minutes to run a one epoch run, and 71 minutes to run a two epoch run.

- **RQ1.1** *Would a BM25-BERT Re-ranker benefit from extra relevant documents missed by BM25?*

This paragraph describes two ways of enriching the training sets with the missed documents:

- *adding* the missed relevant documents to the training sets, this method is referred to as BM25@100-add.
- *swapping* the missed relevant documents with the lowest-ranked BM25 results and replacing them with missed documents[2], this method is referred to as BM25@100-swap.

There are some remarks with both methods, namely when adding documents, resulting in lists of documents relevant to a question becoming longer - on average, three documents. Which is 3 %, therefore it is considered acceptable.

The potential problem with swapping it that a relevant document, found by BM25, is swapped with a missed document. This is not expected to happen, because only the lowest ranking document are candidate for a swap. A check proves this expectation to be correct: over all 255 BM25 ranking (one for each question) a total of 683 documents were swapped in, at a cost of 10 relevant documents.

Together with the baseline (referred to as BM25@100 in the table), the results of various combinations are presented in Table 2.1. The results indicate that the baseline outperforms the other training sets. This is an interesting observation, because the general idea is that more training data is better, although also seen as a paradox. In the case of adding data, there is indeed more training data, but when swapping, the size of the training data is kept the same. Nonetheless, in both cases, adding documents that do not fit in the BM25 algorithm does not positively influence the performance. It would be interesting to discover a cause for this; one could argue that the additional documents contain features that may steer the model away from a certain optimum.

Furthermore, only in the case of swapping, using a learning rate of 3e−5, running an extra epoch improves the performance.

2.3 Inductive Transfer Learning

Instead of enriching the training data, this paragraph proposes inductive transfer learning to enrich the model. This is done by training the model first for a different task with

[2]Because the logits of the query-document combinations are sorted afterward, is does not matter in what order the missed documents are added.

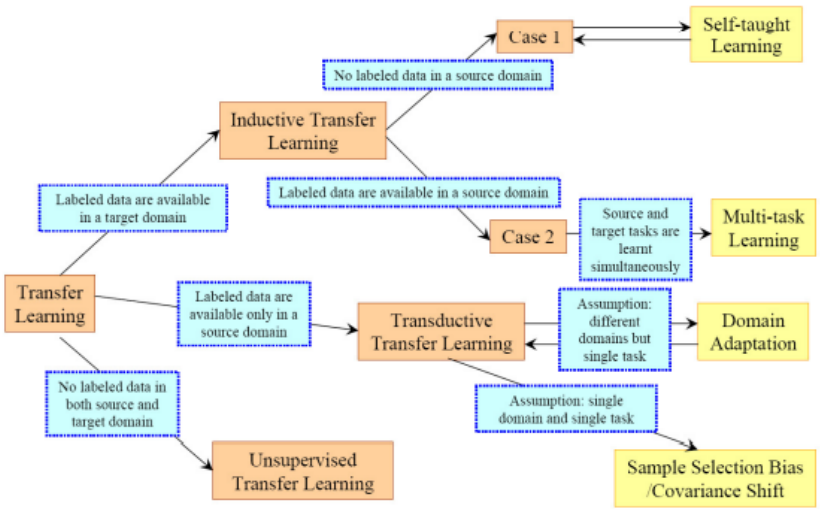

Figure 2.1: The overview of types of transfer learning [9].

the same data, followed by a training for ranking; this method is also called inductive transfer learning.

A general benefit of transfer learning is that the model has not to be trained again for a new task or a similar dataset, but just a small fraction. This could save time and resources [9]. In the case of BERT, instead of e.g. a final layer, all parameters are trained, but only for a few epochs. Besides this, when little data is available it could be the only way to achieve better results. Inductive transfer learning is a specific case of transfer learning, Pan, and Yang (2010) illustrate this nicely in Figure 2.1. The Cranfield data set has all the labels both for training and testing. The general idea with machine learning models is that they mostly need a huge number of labeled examples. However, transfer learning theory states that having learned a task extensively may be easier to learn a closely related task. The tasks used in this research are the Masked Language Task (MLM) and Next Sentence Prediction (NSP).

The research question of interest here are:

- **RQ2.1** *Would a BM25-BERT Re-ranker benefit from inductive transfer learning with Masked Language Modelling?*
- **RQ2.2** *Would a BM25-BERT Re-ranker benefit from inductive transfer learning with Next Sentence Prediction?*

2.4 Transfer Learning with Masked Language Modelling

Instead of enriching the training data, this section proposes inductive transfer learning to enrich the model by training it first on a different task. This is done by first training the encoder of the model for the Masked Language Modelling task, followed by a training step for the ranking task.

MLM Integration in the Re-ranker

The MLM task itself is inspired by the Cloze task[18]. The goal is to guess masked words or tokens in a sentence. It requires, therefore, a certain knowledge of the sentences and documents. An illustration later on in this section makes this clear. The remainder of this section will introduce how this task is incorporated in the pipeline, followed by a description of the MLM-training task itself.

Inductive Transfer learning with MLM The MLM task is integrated with a baseline version of the re-ranker, so no other changes must be considered. First, a pre-trained `bert-base-uncased` default model is trained for the MLM task; the next paragraph discusses these training details. The resulting model is used for inductive transfer learning, during which it is trained for the ranking tasks. The experiment will focus on the influence of learning rate, the number of epochs, and any configuration setting of the MLM training.

One technical remark here: loading the BERT model is not plug and play as the final layer for MLM is not of the same size as the one required for ranking. MLM has an output size equal to the token dictionary, where the ranker only has two outputs, one for relevant and the second for not relevant. Therefore only the encoder is loaded. More details can be found in the repository [3].

MLM Implementation How is the MLM task implemented? The general algorithm used is presented in pseudo-code in Algorithm 1, which shows how it tokenizes 15% of all tokens. This percentage is based on the BERT pre-training, which uses a more complicated method, but takes the 15% as a start[4]. As an example, the following snippet gives an illustration of the masking and demonstrates the effect of 'consecutive masking':

> *"[CLS] the un [MASK]y lift of a wing [MASK] finite aspect ratio [MASK] [MASK] [MASK] - lift functions [MASK] wings of finite [MASK] ratio have been calculated by ..."*

The sentences demonstrate that at some points, a sequence of tokens is masked. For humans, even with domain knowledge, it is already hard to guess what is masked here, let alone an algorithm; BERT predicted the following:

> *"[CLS] the unsteady lift of a wing of finite aspect ratio the the initial - lift functions of wings of finite aspect ratio have been calculated by ..."*

Which, for illustration purposes, should have been:

> *"[CLS] the unsteady lift of a wing of finite aspect ratio unsteady - lift functions for wings of finite aspect ratio have been calculated by ..."*

To exclude and inspect the effect of consecutive masking, the algorithm was changed such that it has the ability to not mask any adjacent token. With those two options,

[3] `https://github.com/MauiGonzo/BERT-BM25-Thesis-Project`

learning rate	5e−5	5e−5	5e−5	2e−5
epochs	2	3	2	2
batch size	16	16	32	16
with consecutive masking	0.7267	0.7187	0.7113	0.7107
without consecutive masking	0.7371	0.7370	0.7316	0.7371

Table 2.2: The results of MLM training, all executed with a `max_length=128`. The last row shows the test accuracy, which the percentage of tokens predicted correctly.

the model was trained for two epochs with a train/test fraction of 60 : 40.

Algorithm 1: MLM train and test set construction - MLM-basis

input : Cranfield Corpus
output: Set of tokenized, partly masked, documents

Tokenize(Corpus);
Copy tokenized Corpus to label;
for *document* ***in*** *corpus* **do**
| Randomly mask 15% of the tokens in document
end
Split in train and test set
Train for MLM-task

MLM training and ranking results

This paragraph discusses the accuracy results for training the MLM task and ranking performance after transfer learning. MLM training results can be found in Table 2.2, where it can be seen that without consecutive masking, the model performs around two percent better. Furthermore, the results indicate that the influence of learning rate and epochs performed is very small compared to the consecutive masking option.

The model with the best overall performance was selected for transfer learning with BERT following this training experiment. The hyperparameters of this MLM-trained model are *lr*=2e−5*; epochs=2; batch size=16 and without consecutive masking.* The results of this experiment are shown in Table 2.3. The results indicate that the baseline NDCG@20 performance, reported in Table 2.1, is improved by around 1%.

Experiment Variations While preparing this experiment, the first transfer learning experiment was executed with a MLM pre-trained encoder and a default (random) initialized task-head, the ranking experiments performed with this setting demonstrated that this gives random results, e.g., it gave an NCDG@20 in the order 1e−2. Like any other model, BERT requires randomly initialized layers to be trained before the output anything meaningful. As the encoder is already trained, the manual is called 'training downstream' in the context of the pre-trained BERT with a task head.

To following method omits the problem with the random initialisation: it simply keeps the encoder weights 'frozen', but trains the task head layer, using this setup, experiments been performed. A minor gain in processing time was noted: the number of trained parameters went down from 109M to 24M, the training time required however went down by half. With a frozen encoder, the training time took 25 minutes on a Tesla T4 per epoch, or 47 minutes on a Tesla K80. With a trainable encoder, it takes 71 minutes (Tesla K80). One of the initial experiments used a general setting of `lr=`2e−5 and 1 epoch, the results showed an NDCG which was significantly lower compared to the baseline, despite the processing this performance was deemed too low, therefore this method was no longer pursued.

learning rate	2e−5		3e−5	
epochs	1	2	1	2
baseline	0.5445	0.5482	0.5439	0.5486
MLM-transfer learning	0.5451	**0.5548**	0.5380	0.5524

Table 2.3: NDCG@20 performance after transfer learning with a MLM-trained encoder, compared to the baseline which is trained with a default `bert-base-uncased`. The highest performance is formatted in boldface. The used MLM-trained encoder hyperparameters can be found in Table 2.2, first column, with the option 'no consecutive masking'.

2.5 Transfer Learning with Next Sentence Prediction

The previous section described how MLM was used to enrich the model. This section describes how the same idea is applied using Next Sentence Prediction (NSP). The pre-trained BERT model is first trained for the NSP task, followed by a training step for the ranking task. The following paragraphs will give a more detailed description of NSP-training integration in the re-ranker, followed by a description of the NSP training itself. The last section will give the results and analyze them.

2.5.1 NSP Integration in the Re-Ranker

The NSP has the objective to tell whether or not a sentence a follows sentence b. In fact, BERT will predict how sure the model is about the correctness of this a-b-pair being the next sentence. The training section will describe how different a-b-pairs can be constructed from the same corpus. Also, the integration of NSP for inductive transfer learning is discussed.

2.5.2 Transfer learning with NSP

This integration very closely resembles the integration of MLM; in fact, there should not be any difference. In this case, transfer learning with NSP is integrated with a baseline version of the re-ranker. Furthermore, every instance of the NSP-trained model is based on a fresh `bert-base-uncased` pre-trained model. The resulting NSP-trained model is used every fold; i.e., when a NSP-model is trained with method M all folds start with this same model.

2.5.3 NSP Implementation

This paragraph and the following one will describe how the Next Sentence Prediction training is organized. The typical input for this task is as follows: *[CLS] sentence a [SEP] sentence b [CLS]*. Apart from the BERT hyperparameters, such as learning rate and batch size. The construction of the a-b-pairs can vary, enabling various training sets. The following describes the construction of the various sets. All the construction methods are based on the same idea, which is represented in Algorithm 2. Note that this algorithm ensures that there will be an incorrect pair for every correct pair, therefore making the data set balanced. The results are presented in Table 2.4. In the leftmost column, the various training sets are named, which will be discussed below.

Dataset based on random sentences This method takes of every document in the corpus (using a for loop) a random sentence, not being the last sentence, and calls this sentence a. With a 50% chance, the model will pick the next sentence or a sentence

from the bag of sentences following this method. Since there are 1400 documents in the corpus, it creates an equal number of sentence pairs. The input for this task hence becomes: *[CLS] document sentence a [SEP] (random) sentence b [CLS]*

Algorithm 2: Generic NSP train set construction.

```
input  : Cranfield Corpus
output: Two same size sets of sentences, that have a 50% change of being next
        sentence
construct empty list A;
construct empty list B;
for document in corpus do
    for Sentence in document do
        bag-of-sentences ← Sentence
    end
end
for document in corpus do
    A ← Sentence from document
end
if random.random() > 0.5 then
    B ← RandomItem(bag-of-sentences)
else
    B ← next Sentence
end
Split A, B in train and test set
```

Dataset based on all sentences Where the previous paragraph only selected one sentence from a document, this method takes every sentence of a document. This way, more pairs can be generated: almost 4000. Only every odd sentence is selected as sentence *a*, this is done to prevent re-using the same sentence for either sentence *a* and a step later (with 50% chance) as sentence *b*.

Dataset based on title sentences and one random Observing that every document has a title, this a-b-pair creation method exploits the fact that a (good) title sentence is remarkable in a way that it relates strongly to most sentences in a (smaller) document. Therefore this dataset is constructed as follows: from every document, it takes the title as sentence *a* and either a random sentence from the same document or a random sentence from the bag of sentences as sentence *b*.

Dataset based on title sentences and all sentences Building on the idea described in the previous paragraph, this model considers that the title relates to all sentences in the documents. Therefore all combinations of title sentence and document sentence are constructed. Many title sentences are also paired with a sentence from the bag of sentences to balance the set. This method results in the largest dataset, counting almost 2e3 pairs.

2.6 NSP Ranking Results

This section presents the results of the experiments, each using a different training set for the NSP training. The performance results can be found in Table 2.4. For time and

learning rate	2e−5		3e−5	
epochs	1	2	1	2
baseline	0.5445	0.5482	0.5439	**0.5486**
MLM-transfer learning	0.5451	**0.5548**	0.5380	0.5524
NSP random-sentence	0.5403	0.5386	0.5427	0.5450
NSP all-sentence	0.5478	0.5494	0.5316	0.5341
NSP title+50/50	0.5478	0.5420	0.5284	0.5402
NSP title+all doc	0.5505	**0.5587**	0.5480	0.5292

Table 2.4: The re-ranker NDCG@20 performance when applying inductive transfer learning, using various MLM- and NSP-trained models. For each category the best performance is formatted in boldface.

computation resources reasons, not all combinations are tested, only promising leads were followed. For instance, the frozen encoders were left out in the runs with the titles since this lead performed at least 10% lower during initial testing.

Training a random-sentence dataset takes around 1 to 2 minutes on a Tesla T4. The exception is the 'title+all doc,' which took 7 minutes to train; this can be explained by the dataset size, which consists of almost 20.000 sentence pairs. Using this dataset in NSP-transfer learning leads to the best performance; even with one training epoch, it already outperforms the baseline. The best performance outperforms the NDCG-performance by almost 2%. Furthermore, it can be observed that training for one epoch achieves better performance than training for two.

Further training the models in the re-ranker took in all one epoch cases 37 min, and in the case of two epochs 70 min, both using a Tesla T4 on GCP.

2.6.1 NSP Results Analysis

The performance figures are given by Table 2.4. The bold figure indicates the overall best performance of all the experiments in this table. The first row gives the baseline performance, followed with the best MLM-performance. Focusing on NSP, a number of observations can be derived from this overview: the learning rate of 2e−5 may lead to better performance, furthermore, training an extra epoch does not always lead to better performance. Looking at the last, best performing, row it is clear that even after training for one epoch the performance is already outperforming the baseline. So, resource-wise this is a big gain, because at the cost of the NSP-training (which is around 1/5 of a regular epoch) a whole epoch could be omitted. The conclusion here is that the use of inductive transfer learning with NSP is beneficial for a re-ranker, both in performance and resource consumption.

When taking into account the MLM-performance it can be observed that although the MLM-training requires less time, compared to NSP, the one epoch performance is outperformed by NSP transfer learning. Although the final performance is almost as good as NSP, here one could argue that, when top performance is paramount, the overall resource/performance ratio could be in favor of MLM.

Variations and further testing During initial testing some leads have been tried and abandoned since they proved not very fruitful. Nonetheless they are considered worth mentioning. Among them are:

- the question whether more training with NSP will improve the final ranking score, for this reason a NSP was trained with two epochs using the all-sentences: this resulted in a NDCG@20 of 0.5398, which not better than any other method used.

- the question whether freezing the encoder parameters helps (during training with the re-ranker). Test revealed that the NDCG@20 performance was at best 0.5135 (in the case of random-sentence, with two epochs). While running it in e.g. with the random-sentence trained NSP-model the performance after one epoch was 0.4962, where the reduction in computation time was around 5 minutes.
- the question of using 'sub-sentences': Since the average length of a document is around 1000 words, and the fact that BERT 'does not care' whether a sentence starts with a capital and ends with a period. The documents could be split in sub-sentences of n words. This was tested with $n = 30$, but the NDCG@20 performance after one epoch was around 0.52. It seems BERT does care a little whether a sentence is logically.

Chapter 3

Multi Label Ranking

3.1 Introduction

Where Chapter 2 relies on binary relevance labels, this Chapter presents methods that rely on multiple relevance labels. The Cranfield Corpus provides multiple relevance scores for relevant documents, ranging from 1 to 4, where 1 is 'clear answer' and 4 indicates 'minimum interest' and 0 indicates no relevance to the question. This section will describe a method to research whether the relevant levels of the query-document combinations can be leveraged for the BM25-BERT re-ranker. After a brief introduction on relevance levels, training using customized task heads is described, followed by testing methods.

3.2 Multi Label Integration in the Re-Ranker

The following section describes how the basic re-ranking pipeline is modified to leverage multiple relevance levels. Several methods have been integrated; the details will be described in the following paragraphs. The methods can be split into training the model and ranking the results after testing them with the standard test set. The training methods combine a set of hyperparameters and several methods for deriving the training loss. This thesis refers to the methods by the name of their loss. To introduce the idea: the first and most straightforward method is to give the model five labels as the target, change the final layer such that it has five outputs, and change the loss accordingly. This allows training with multiple relevance levels. The default BERT task head for ranking provides this option, more advanced methods require custom task heads, which the following sections will describe. The evaluation methods describe how to handle the predictions because, after testing, the model predicts each label, i.e., a logit for label 0...4. In the default case, the results contained only two predictions: relevant and not relevant. Hence ranking was based on the scores for label "1" (relevant). Now there are five predictions; the most straightforward way is to select the best, i.e., taking the $arg\,max$, this will indicate the label, and the $a\,max$ will give the score. Each document in the batch is now classified and has a score. Based on these parameters, a ranking can be made. Besides the $arg\,max$-method, this thesis researches other methods to decide what predictions to choose. Due to the nature of the resulting predictions, an alternative method has been tested. This method considers only one label; this method is referred to as the 'per label approach'.

Binary Cross Entropy Loss

The default task head for ranking, provided by Huggingface, has per default the option to set the number of labels. When the number is larger than one, the class will use a loss that is a combination of a Sigmoid layer and a Binary Cross-Entropy Loss [1]. By default, the class weights are all the same (that is 1). However, the data is not balanced; only a small fraction of the documents are classified as relevant to a question. An approach to counter the class imbalance is applying weights to the losses. Therefore the factions for each label are counted, and this results in a weight vector, i.e., the actual imbalance of class 1,2 and 4 is around 1/50 and 3 is 1/20.

Table 3.1 presents the results after training with either the default BCEwithLogitsLoss and the one with a weight vector $[1.0, 50.0, 50.0, 20.0, 50.0]$.

arg max and 'per label' based ranking

After training, the model is tested. After inputting a test set, the model returns predictions. In this case, the model has multiple outputs, resulting in a prediction for each label. A common approach to evaluating a classifier model's multiple outputs is taking the node with the highest probability. This straightforward approach is applied to classify the best performing relevance label or class for each query-document combination. However, since this is a re-ranker solving a ranking problem, the algorithm is a bit more complicated than just classifying an input. It has to rank a batch of inputs; the results can be found in Table 3.1. The following enumeration describes the steps required to rank the results:

1. use the *arg max* to assign a class;
2. use the *amax* to assign a document score, which is the logit for this label;
3. sort the batch for class and document score, note that the values for label 0 have to be sorted the other way around because they indicate irrelevance;
4. arrange the batch such that the documents with the highest relevance and the best document score are first, and last is label 0 with a high score, which indicates most probable not relevant.
5. use this ranking as input for the evaluation metrics such as MAP and NDCG.

On closer inspection, however, it turns out that the predictions show bias. For every document evaluated, the inspected batches show that all predictions n for class c follow a certain pattern, this could e.g. be: $\forall n \in \text{batch} : \hat{y}_{n,c=2} > \hat{y}_{n,c=3} > ... > \hat{y}_{n,c=1}$, therefore in this example class 2 is always the max logit.

Therefore a different ranking method is tested, which closely resembles the original implementation, only this time all the labels have been tried. Given a label, it takes all the scores and uses them to rank the documents; when doing this for all the labels, it will give the result shown in Table 8.

3.3 Results for Single Class, Multi Label Loss

The following section presents and analyses the results found when applying the methods presented in the previous section. The results for the various losses evaluated with the *arg max* are presented by Table 3.1. Table 3.2 presents the results with a per label

[1] The class uses the Pytorch `BCEwithLogitsLoss`, `https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html`, last visit on 9/29/2021

learning rate	2e−5		3e−5	
epochs	1	2	1	2
baseline	0.5445	0.5482	0.5439	**0.5486**
BCEwithLogitsLoss (default)	0.5348	**0.5448**	0.5332	0.5419
BCEwithLogitsLoss (custom)	0.5188	0.5123	0.5182	0.5112

Table 3.1: The re-ranker NDCG@20 performance when leveraging the multiple relevance levels during training and ranking them on a method that uses the *arg max* of the predictions. The first row is the baseline, with no alterations. The second row indicates what the influence is of training with multiple labels, the third indicates what the effect is of setting class-weights.

learning rate	2e−5		3e−5	
epochs	1	2	1	2
baseline	0.5445	0.5482	0.5439	0.5486
label 0	**0.5440**	**0.5444**	**0.5372**	**0.5446**
label 1	0.5394	0.5437	0.5237	0.5400
label 2	0.5101	0.5431	0.5090	0.5353
label 3	0.5024	0.5141	0.5131	0.5273
label 4	0.4651	0.5089	0.4101	0.4908

Table 3.2: The re-ranker NDCG@20 performance when leveraging the multiple relevance levels during training and ranking them on the scores found for a particular label. Every row indicates the performance when using that rows label. Every run the loss derived with the default BCEwithLogitsLoss. The best performing label is formated bold for easy comparisson with the baseline.

performance, based on a loss that is calculated with the Binary Cross-Entropy method with default equal unit weights, as described above.

Based on the results in Table 3.1 is clear that the class-weights assigned had a negative effect on the performance. When comparing the other results with the baseline, is it clear that this setup yields similar results. When looking at the other ranking method results in Table 3.2 it is clear that label 0 is the best indicator, followed by labels 1 and 2. For each combination of hyperparameters, label 0 returns the best performance, which closely resembles the baseline and even closer to the *arg max* results. The final observation can be explained by the fact that the predictions show a bias, i.e., when the predictions for label 0 are the highest, the ranking is per label 0.

The run times of the results for the *arg max* method take in general about 40 minutes for one epoch and 75 min for two epochs. The same was observed for the per-label approach. The overall benefit of two instead of one epoch is a gain of around 1e−5 (in NDCG performance), the resources required to achieve this performance are considered not worth the gain.

3.4 Multi-Class, Multi-Label Loss Integration

This section describes an approach based on the idea that the classification problem can be seen as a multi-class, multi-label problem. Here we have two relevant or not classes, and within the relevant class, it has a relevance label. This idea is also referred to as *multi task* [14, 24]. The implementation of this idea is similar to that of Single Class, Multi-label. This method can be seen as an extended version because it creates a final layer of two nodes (for the class) and another final layer of five nodes (the relevance

learning rate	2e−5		3e−5	
epochs	1	2	1	2
baseline	0.5445	0.5482	0.5439	0.5486
mutlitask BCEwLL 50/50	0.5324	0.5420	0.5262	0.5430
mutlitask BCEwLL 80/20	0.5224	0.5370	0.5339	**0.5469**

Table 3.3: The re-ranker NDCG@20 performance when leveraging the multiple relevance levels during training. In this experiment the model is trained as Multi Class, Multi Label, i.e. with two losses. In the second row the losses are averaged, and in the last row they are combined 80/20. The ranking is based on the *arg max*.

levels). Both final layers are connected to their loss function, and therefore they can be trained on a specific target. The loss that is fed back to the model is comprised of the class-loss and the label-loss. The most straightforward method is averaging them, but one can also do some manual or automatic tuning. A loss ratio of 50-50 and 80-20 (class-loss versus label-loss) is used in the experiments. The latter ratio is based on the idea that there is more data in the case of classifying for relevant or not. As can be seen in Table 3.2, labels 3 and 4 are poor indicators. Each loss uses the earlier method: a Sigmoid layer connected to a Binary Cross-Entropy loss, where no specific class weights are set. The final ranking is achieved using the *arg max*, in the same fashion as described in the sections above. The loss-combinations are tested against the various combination of hyperparameters; the detailed results are presented in Table 3.3.

3.5 Results for Multi Class, Multi Label

This section presents the results after training the model using a multi-class, multi-label method, also known as multi-task. The results are presented in Table 3.3, in general, the wall time required for training was about 40 min for one epoch and 74 min for two epochs. It can be observed that the 80/20 ratio performs almost as good as the baseline when trained with a learning rate of 3e−5 and two epochs. Overall it can be concluded that the multitask approach is capable to produce near-baseline performance. However this method does not seem to leverage the extra information given.

3.6 Regression Implementation,

All the previous methods used BERT as a classifier; given an input, the BERT output was the most likely class. This section explains a method that translates the classification problem into a regression problem. The following will first give a little background in general and elaborate on the BERT case in particular before describing the used method with various options in detail.

3.6.1 Regression With BERT

Remember the goal of the ranking pipeline, retrieve the best ranking. For BERT, this is ranking a set of N documents found by BM25@N given a query. Therefore, BERT requires that it tells how to rank a document; for this, the model only has to assign a score to each document, where the highest score would be the most relevant and vice versa. When the ranking problem is treated as regression, this will be the model's expected outcome. It will require the task head to have one output, returning the ranking score, and the model requires targets during training. The relevance levels can

be used for this, but conversion is required since they are not between -1 and 1. Besides, the relevance grading system is not continuous, which is required for regression.

The setup is based on the baseline; the changes are as follows: the task head only has one output, the document relevance for a query $\hat{y}_{dq}$; the targets are a conversion of the annotated relevance score. The experiments with the method focus on the various conversion tables. First, several conversions have been tried with one set of hyperparameters to assess which are promising; the results of this can be found in Table 3.4. Furthermore, one special case is exactly the baseline setting, i.e., not relevant has targeted $y = 0$, and each relevant case gets $y = 1$. Following the initial experiment, a number of conversions are tested for the combinations of hyperparameters used in the other experiments; the resulting NDCG-performance can be found in Table 3.4.

A final remark, before moving on to the implementation and results, is the motivation to have a target $y \in (-1, 1)$ is that when the range goes up to e.g. 10.000 the smaller numbers lose importance, e.g. the difference between 5 and 6 can become insignificant. This effect is often demonstrated and countered in various blogs using the Boston Housing Dataset [6], where the house prices vary between $1000 and $1.000.000.

3.6.2 Regression Implementation

Implementing regression for ranking in BERT requires only minor changes concerning the baseline. The output layer must only contain one node; consequentially, the Huggingface task head class switches to Mean Square Error (MSE) loss. The targets do technically not require conversion, but the multi-relevance labels are sorted in ascending order and converted for better performance. Since only one node generates output, after one batch, the list of logits contains one ranking score per document; this value is evaluated as it would in the baseline.

The main factor that can and will be varied is the conversion used. Besides the usual hyperparameters learning rate and epochs, no other setting is changed. The following section provides more details on the experiments.

3.6.3 Regression Test Results

The section presents and analyses the results of the experiments introduced in the previous section. Contrary to earlier experiments first a small feasibility study has been done, presented in Table 3.4, the hyperparameters here are *lr*=1e−5 and *epochs=1*. The goal was to find which conversions might be promising and to test the hypothesis regarding larger numbers. Following this, the usual extensive setup is applied: vary the method at hand combined with various combinations of the hyperparameters *learning rate* and *epochs*. Table 3.5 presents the results of this setup, the next paragraph explains the selection made.

The results in table 3.4 indicate not all conversions are promising: using relatively large values corresponds with relative low NDCG-performance. Furthermore, converting the label 0 to either 0 or −1 does not matter when the other values are converted to 1. To keep the use of resources within limits, the selection for the full experimental setup consist of the following conversion tables:

- The one that reflects the setting that is similar to the baseline;
- The two most promising, because they have the best NDCG.

The results after testing the selected conversion tables with all hyperparameter settings of interest can be found in Table 3.5. Every run with one epoch took 39 minutes (plus or minus 1 minute), and the runs requiring two took epochs around 75 minutes

conversion					*score*
0	1	2	3	4	NDCG
0	1	1	1	1	0.5389
1	10	8	6	5	0.5355
0	50	45	40	35	0.5144
−0.5	1	0.8	0.6	0.3	0.5424
−1	1	1	1	1	0.5389
−1	1	0.9	0.8	0.1	0.5388
−1	1	0.7	0.53	0.4	0.5420

Table 3.4: Table representing the results when running BERT in regression mode. The rows reflect the effect of various conversions. The hyperparameters here are lr=2e−5 and *epochs=1*. The first row reflects the baseline relevance i.e. zero indicates not relevant, one is most relevant down to 4 which is the lowest relevance score. The conversion translates this to a ascending scale . The final row is log-translation: given the score s it assigns relevance as follows: rel $= 1 - log(s)$.

learning rate					2e−5		3e−5	
epoch					1	2	1	2
conversion								
0	1	2	3	4				
0	1	1	1	1	0.5389	0.5432	0.5306	0.5495
-0.5	1	0.8	0.6	0.3	0.5424	**0.5527**	0.5358	0.5462
-1	1	0.7	0.53	0.4	0.5420	0.5481	0.5350	0.5502
baseline					0.5445	0.5482	0.5439	0.5486

Table 3.5: Results after applying the regression-method on the 'baseline' setting and most promising settings found in Table 3.4. The 'baseline' conversion containing only zero and ones is in fact a representation of the baseline, only treated differently.

(plus or minus 5 minutes). The first observation is that any conversion leads to the best performance that surpasses the baseline. The best improvement is 0.73% (or 0.0041) in NDCG-performance. Similar performance was only achieved when using a max length of 256 [5]. Another observation is that there is not one 'best' learning rate.

Why performs this method at least as good as the baseline? Contrary to the method previously described, which was based on the *arg max*, no choice for output to evaluate had to be made. When a choice is made between multiple outputs, other predictive information is left out. This information is combined by the final layer, using regression, which is the better option of the two methods. The section following this will elaborate on attempts to improve the result by simple rule-based combining predictions.

3.6.4 Feedback of Threshold, BM25-tuning and Dropout Tests

This section describes some methods that have been tried and tested to enhance or improve methods. At first hand, they did not seem promising. Therefore these leads have been abandoned. Nonetheless, the efforts will be discussed briefly for reasons of completeness and sketch a context for the effectiveness of the approaches tested extensively.

Thresholding

All improvement methods discussed are not tested extensively; therefore, the results are left out. For instance, testing ceased after receiving very low initial performance numbers; such results do not offer an easy comparison. A challenge with the multi-label setup is to evaluate the various label predictions. The approach that uses $arg\,max$ nor taking one label returns results that are better than the baseline. One of the causes is the bias towards label 0 because there is class imbalance. One idea is to threshold the results such that 10% of the results are chosen from labels 1-4, while the other 90% is based on label 0's score. An initial test indicated that this approach was not beneficial.

BM25-tuning

Other ideas focus on reducing the class imbalance: therefore, BERT has to be trained an equal amount of relevant and not relevant documents. A simple method is reducing the results of BM25@N; this was tried with $N = 30$ but did not prove beneficial.

Dropout

The subject of multi label prediction with BERT is discussed in various blogs [2] [3] [4] , it was observed that the ones reviewed did not mention the use of a Dropout layer. For this reason various task heads without a Dropout layer have been tested, those were the following:

- The baseline approach, but with a custom task head without Dropout layer, the performance was comparable with the baseline.
- The multi-label, multi-loss approach was also tested in a configuration without Dropout; however, this run was done with a Cross-Entropy Loss. The results here indicated that this approach was not beneficial, but since the BCE with Logits Loss was not used here, this may also be a cause.

3.6.5 Analysis

The results for the regression based method given in Tables 3.4 and 3.5 lead to a number of observations. First of all, the results indicate that the hypothesis about using a conversion where $score > 1$ seems to be correct, because the performance is poor compared to the baseline. Furthermore it seems that the actual small changes in the conversion make quite a difference, when comparing the result with the baseline. This baseline is interestingly different from the conversion that resembles the 'real' baseline. However there is one difference which is the loss function, the 'real' baseline uses `BCEwithLogits`. Given that it is the only difference one can conclude that changing the loss might be a very interesting topic for further research. This brings us to the best result so far, which is beyond baseline performance, leaving the conclusion that regression is a beneficial method.

Zooming out, the following paragraph analyses the results for both regression and multi-class loss, the results of the latter can be found in Tables 3.1,3.2 and 3.3. Overall, the multi label method in any variant did at best almost achieve baseline performance,

[2] `https://medium.com/analytics-vidhya/multi-label-text-classification-using-transformers-bert-93460838e62b`, last visit on October 5th, 2021

[3] `https://curiousily.com/posts/multi-label-text-classification-with-bert-and-pytorch-lightning/`, last visit on October 5th, 2021

[4] `https://towardsdatascience.com/transformers-for-multilabel-classification-71a1a0daf5e1`, last visit on October th 2021

where regression performed fair i.e. it was beyond baseline. Since the differences between the two methods are not singular it is not possible to derive a conclusion. The main differences however are the type of loss, and therefore the method how the ranking is determined i.e. the $arg\,max$ method. Possibly, regression could be trained with various types of loss, as described in the previous paragraph, that may give a broader view on the topic. Nonetheless one could conclude that the $arg\,max$-way of ranking is outranked by a neural method.

This last analysing paragraph will discuss the section of methods that did not prove promising. The thresholding method was not fruitful, just like $arg\,max$ it is a rule based method, and is therefore easily outperformed by a neural method. Next was the idea of balancing the dataset more by reducing the N in the BM25 pre-ranking step, this did not yield any interesting results. A possible cause might be that this yielded too little training data. The final method tried was the absence of dropout in the task head. As described the loss differs from the 'regular' loss, so a fair comparison here is hard to make. The method tried gave baseline performance, however there is still the $arg\,max$-way of ranking. Possibly, removing dropout from a proven method such as transfer learning, might be very interesting.

Chapter 4

Conclusion

This part answers the research questions and presents some observations that may lead to new research questions.

4.1 Research Questions

This thesis has investigated various methods to fine-tune BERT in a BM25-BERT re-ranker setting. For this, various experiments were constructed in which different parts of the training pipeline were adapted. The research questions leading to these adaptions are reviewed below. Which, in conclusion, can be answered as follows:

- **RQ 1.1** - *Would it be beneficial to enrich the BM25-results with 'missed' relevant documents?*

  The baseline outperforms all the experiments conducted. Therefore this method is not deemed beneficial.

- **RQ 1.2** - *Would it be beneficial to fine-tune BERT on another task, such as Masked Language Modeling (MLM) - also known as inductive transfer learning?*

  Inductive transfer learning with MLM outperforms the baseline in any case of two epochs, with a NDCG@20 performance of 0.5548. Besides, reaching the baseline performance can be done with significant resources. Therefore this method is deemed beneficial.

- **RQ 1.3** - *Would it be beneficial to fine-tune BERT on another task, such as Next Sentence Prediction (NSP) - also known as inductive transfer learning?*

  Inductive transfer learning with NSP trained on a *title+docs* dataset outperforms the baseline and MLM-method of RQ 1.2 with a NDCG@20 score of 0.5587. Moreover, just as in that question, this case let one reach a beyond baseline score with fewer resources than the baseline would require. This method is therefore deemed most beneficial.

- **RQ 2.1** - *Would it be beneficial to train BERT with single class, multi label targets?*

  Even after two epochs, this method is not able to outperform the baseline. Therefore this method is not deemed beneficial.

- **RQ 2.2** - *Would it be beneficial to train BERT with multi class, multi label targets?*

  After two epochs, this method is almost as good as the baseline. This is not considered beneficial.

- **RQ 2.3** - *Would it be beneficial to train BERT on the relevance labels level directly, in other words would regression be beneficial?*

  This method is implemented as regression, which after two epochs of training proves to outperform the baseline method with an NDCG@20 score of 0.5527. Therefore this method is deemed beneficial

All in all, it can be concluded that inductive transfer learning is the most beneficial of the methods investigated. Also, it can be concluded that when training for a different task, with the same data, the method and the way of training the other task have a significant influence. Furthermore, it can also be concluded that training with the regression method is a promising method. The benefit here is that the neural network predicts only one ranking; in other setups, multiple rankings were predicted, which had to be filtered (e.g., $arg\,max$ filtering, or using one label). Rule base filtering has been replaced by regression, which leverages the abilities of the more complex neural network.

4.2 Future Leads

This section presents an overview of leads for future research. Since a huge constraint in this type of research is the resources available, several leads include executing more experiments. This thesis already required over 80 runs, with an average running time of almost 1 hour per run (using a NVIDIA Tesla T4). The takeaway here is to experiment with promising leads, heed the not promising ones and conduct a fine strategy before scaling things. The following paragraphs will discuss some leads that apply to those methods; afterward, some general leads are discussed.

Starting with Chapter 2, there is a small lead that is extending the work that used NSP as a task to train; the rationale here is that it seems to matter how the NSP task is trained. Possibly there might be variations here.

Following, looking at Chapter 3, a lead here is to test this with other losses both with multitask and regression. An implementation of Hingeloss proved not promising, but maybe others or different implementations. The multitask implementation proved to be comparable with the baseline, so for instance, (manual) tuning the ratio might prove interesting. The final results require a form of rule-based filtering, which the regression method has not. What stands out with regression is sensitivity regarding the conversion table. It may be interesting to look into this; one way is testing more conversions. Another approach is applying a method called *stacked regression* in which multiple regression models are trained and combined. Although resource-demanding, this may be interesting[1]. Besides this, a few runs without a dropout layer have been performed; it might prove interesting to pursue this lead further, by setting the dropout to 0 in promising set ups.

Furthermore, combining the best methods is proposed as lead, e.g., combining inductive transfer learning with NSP could be combined with the multi-label ranking methods. Besides this, the hyperparameter search could be stretched; it could be interesting to double the `max_length` parameter. A deeper problem noted is the class imbalance. Leads that focus on how to deal with such sets are also most interesting. A closing remark on the implementation is that various research used Tensor Processing Units (TPUs) instead of GPUs. This architecture might require fewer resources to achieve the same results.

[1] `https://www.kaggle.com/serigne/stacked-regressions-top-4-on-leaderboard` (last visit: 6-10-2021

Bibliography

[1] Andrew Arnold, Ramesh Nallapati, and William W Cohen. A comparative study of methods for transductive transfer learning. In *Seventh IEEE international conference on data mining workshops (ICDMW 2007)*, pages 77–82. IEEE, 2007.

[2] W Bruce Croft, Donald Metzler, and Trevor Strohman. *Search engines: Information retrieval in practice*, volume 520. Addison-Wesley Reading, 2010.

[3] Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, and W Bruce Croft. Neural ranking models with weak supervision. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 65–74, 2017.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] Negin Ghasemi and Djoerd Hiemstra. Bert meets cranfield: Uncovering the properties of full ranking on fully labeled data. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 58–64, 2021.

[6] David Harrison Jr and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[7] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[8] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*, 2019.

[9] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[10] Brian Quanz and Jun Huan. Large margin transductive transfer learning. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1327–1336, 2009.

[11] Phyllis A Richmond. Review of the cranfield project. *American Documentation (pre-1986)*, 14(4):307, 1963.

[12] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[14] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[15] Mark Sanderson and W Bruce Croft. The history of information retrieval research. *Proceedings of the IEEE*, 100(Special Centennial Issue):1444–1451, 2012.

[16] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

[17] Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. *arXiv preprint arXiv:2004.08900*, 2020.

[18] Wilson L Taylor. "cloze procedure": A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[20] E Voorhees. Overview of the trec 2004 robust retrieval track. In *Proceedings of TREC*, volume 13, 2004.

[21] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[22] Lixiao Xie, Zhaohong Deng, Peng Xu, Kup-Sze Choi, and Shitong Wang. Generalized hidden-mapping transductive transfer learning for recognition of epileptic electroencephalogram signals. *IEEE transactions on cybernetics*, 49(6):2200–2214, 2018.

[23] Emine Yilmaz, Nick Craswell, Bhaskar Mitra, and Daniel Campos. On the reliability of test collections for evaluating systems of different types. In *proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2101–2104, 2020.

[24] Yu Zhang and Qiang Yang. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017.

[25] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27, 2015.