KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Science and Engineering, specializing in Machine Learning

Second cycle, 30 credits

Integrating Telecommunications-Specific Language Models into a Trouble Report Retrieval Approach

NATHAN G. BOSCH

Stockholm, Sweden, 2022

Integrating Telecommunications-Specific Language Models into a Trouble Report Retrieval Approach

NATHAN G. BOSCH

Master's Programme, Machine Learning, 120 credits
Date: June 20, 2022

Supervisors: Forough Yaghoubi, Serveh Shalmashi, Kamal Hakimzadeh
Examiner: Amir H. Payberah
School of Electrical Engineering and Computer Science
Host company: Ericsson AB
Swedish title: Integrering av telekommunikationsspecifika språkmodeller i en metod för hämtning av problemrapporter

© 2022 Nathan G. Bosch

Abstract

In the development of large telecommunications systems, it is imperative to identify, report, analyze and, thereafter, resolve both software and hardware faults. This resolution process often relies on written trouble reports (TRs), that contain information about the observed fault and, after analysis, information about why the fault occurred and the decision to resolve the fault. Due to the scale and number of TRs, it is possible that a newly written fault is very similar to previously written faults, e.g., a duplicate fault. In this scenario, it can be beneficial to retrieve similar TRs that have been previously created to aid the resolution process.

Previous work at Ericsson [1], introduced a multi-stage BERT-based approach to retrieve similar TRs given a newly written fault observation. This approach significantly outperformed simpler models like BM25, but suffered from two major challenges: 1) it did not leverage the vast non-task-specific telecommunications data at Ericsson, something that had seen success in other work [2], and 2) the model did not generalize effectively to TRs outside of the telecommunications domain it was trained on.

In this thesis, we 1) investigate three different transfer learning strategies to attain stronger performance on a downstream TR duplicate retrieval task, notably focusing on effectively integrating existing telecommunications-specific language data into the model fine-tuning process, 2) investigate the efficacy of catastrophic forgetting mitigation strategies when fine-tuning the BERT models, and 3) identify how well the models perform on out-of-domain TR data.

We find that integrating existing telecommunications knowledge through the form of a pretrained telecommunications-specific language model into our fine-tuning strategies allows us to outperform a domain adaptation fine-tuning strategy. In addition to this, we find that Elastic Weight Consolidation (EWC) is an effective strategy for mitigating catastrophic forgetting and attaining strong downstream performance on the duplicate TR retrieval task. Finally, we find that the generalizability of models is strong enough to perform reasonably effectively on out-of-domain TR data, indicating that the approaches may be eligible in a real-world deployment.

Keywords

information retrieval, neural ranking, trouble reports, log analysis, natural language processing

Sammanfattning

Vid utvecklingen av stora telekommunikationssystem är det absolut nödvändigt att identifiera, rapportera, analysera och därefter lösa både mjukvaru och hårdvarufel. Denna lösningsprocess bygger ofta på noggrant skrivna felrapporter (TRs), som innehåller information om det observerade felet och, efter analys, information om varför felet uppstod och beslutet att åtgärda felet. På grund av skalan och antalet TR:er är det möjligt att ett nyskrivet fel är mycket likt tidigare skrivna fel, t.ex. ett duplikatfel. I det här scenariot kan det vara mycket fördelaktigt att hämta tidigare skapade, liknande TR:er för att underlätta upplösningsprocessen.

Tidigare arbete på Ericsson [1], introducerade en flerstegs BERT-baserad metod för att hämta liknande TRs givet en nyskriven felobservation. Detta tillvägagångssätt överträffade betydligt enklare modeller som BM-25, men led av två stora utmaningar: 1) det utnyttjade inte den stora icke-uppgiftsspecifika telekommunikationsdatan hos Ericsson, något som hade sett framgång i annat arbete [2], och 2) modellen generaliserades inte effektivt till TR:er utanför den telekommunikationsdomän som den bildades på.

I den här masteruppsatsen undersöker vi 1) tre olika strategier för överföringsinlärning för att uppnå starkare prestanda på en nedströms TR dubbletthämtningsuppgift, varav några fokuserar på att effektivt integrera fintliga telekommunikationsspecifika språkdata i modellfinjusteringsprocessen, 2) undersöker effektiviteten av katastrofala missglömningsreducerande strategier vid finjustering av BERT-modellerna, och 3) identifiera hur väl modellerna presterar på TR-data utanför domänen.

Resultatet är genom att integrera befintlig telekommunikationskunskap i form av en förtränad telekommunikationsspecifik språkmodell i våra finjusteringsstrategier kan vi överträffa en finjusteringsstrategi för domänanpassning. Utöver detta har vi fåt fram att EWC är en effektiv strategi för att mildra katastrofal glömska och uppnå stark nedströmsprestanda på dubbla TR hämtningsuppgiften. Slutligen finner vi att generaliserbarheten av modeller är tillräckligt stark för att prestera någorlunda effektivt på TR-data utanför domänen, vilket indikerar att tillvägagångssätten som beskrivs i denna avhandling kan vara kvalificerade i en verklig implementering.

Nyckelord

informationssökning, neural rangordning, felrapporter, logganalys, naturlig språkbehandling

Acknowledgments

The work presented in this master thesis has been conducted at Ericsson's Global Artificial Intelligence Accelerator (GAIA) department in Stockholm between January and June of 2022.

I would like to thank my supervisors at Ericsson, Forough Yaghoubi and Serveh Shalmashi, for their guidance throughout the master's thesis. I would also like to thank my examiner at KTH, Amir Payberah, for his insight and advice.

I would also like to thank Henrik Holm and Fitsum Gaim from Ericsson for their advice and help when working with the telecommunications-specific language model.

Stockholm, June 2022
Nathan G. Bosch

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

BE	Bi-Encoder
CE	Cross-Encoder
DA	Domain Adaptation
EWC	Elastic Weight Consolidation
MS	Multi-Stage Fine-Tuning
MS w/ EWC	Multi-Stage Fine-Tuning with Elastic Weight Consolidation
SL	Sequential Learning
TR	Trouble Report

Chapter 1

Introduction

1.1 Premise and Problem Overview

In the development of large telecommunications systems, it is imperative to identify, report, analyze and, thereafter, resolve both software and hardware faults. Cataloguing and sharing these faults is often done through trouble reports (TRs) [3]. At Ericsson, TRs are manually written observations of a software or hardware fault, which after a series of analyses and corrections, are answered and resolved. This process of reaching an answer/resolution at Ericsson, outlined in Figure 1.1, is often tedious and time consuming. In addition to this, as TRs are leveraged by nearly all operators, there is also a risk that duplicate TRs emerge when several developers identify the same fault independently. Resolving each duplicate TR independently leads to a substantial amount of additional manual effort, which may be avoidable if a strong duplicate detection were to be implemented. The focus of this thesis, therefore, is to develop a duplicate TR retrieval system that, given a newly written observation of a fault, can retrieve potential duplicates effectively.

Previous master's thesis work at Ericsson [1] investigated approaches to reduce the latency in the resolution process of TRs, focusing on aiding the Analysis Phase (Step 4 in Figure 1.1) by retrieving previously resolved TRs representing similar faults (not necessarily duplicates). The retrieval of similar TRs could, thereafter, be used to identify if an answer associated with the fault already exists, within which domain the problem lies (i.e., to help identify a team to run further analysis), and generally provides additional information which can help speed up the resolution process. Although duplicate TR retrieval was not the central aim of this previous work, the modelling approach has seen success in retrieving duplicates [1, 4] and acts as the foundation for the retrieval approach in this thesis.

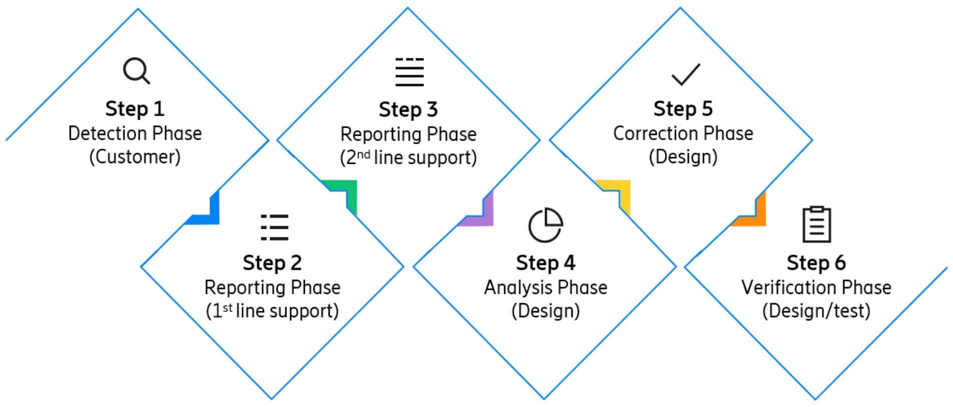


Figure 1.1: Process of handling Trouble Reports at Ericsson

The TR retrieval approach in [1], outlined in Figure 1.2, leveraged two types of BERT (Bidirectional Encoder Representations from Transformers) [5] models to perform efficient and effective retrieval. Given a newly written TR, a faster bi-encoder model would identify the top 20 most similar existing TRs. After which, these TRs would be re-ranked in relevance by a slower, but higher performance cross-encoder model. Note that BERT models and the both bi-encoder and cross-encoder architectures will be further discussed in section 2.2.2.

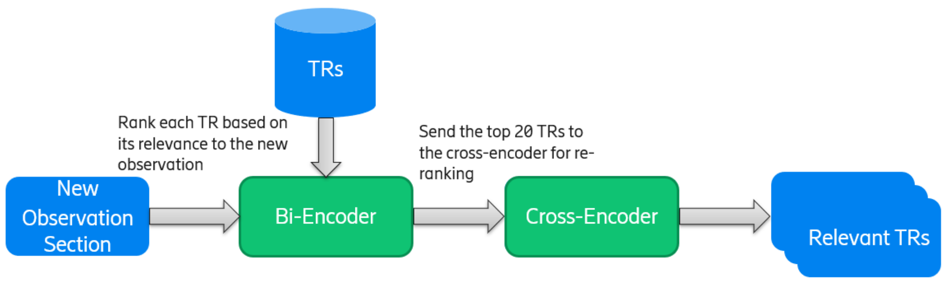

TRs
Rank each TR based on its relevance to the new observation
Send the top 20 TRs to the cross-encoder for re-ranking
New Observation Section
Bi-Encoder
Cross-Encoder
Relevant TRs

Figure 1.2: Multi-stage retrieval process of relevant TRs. Note that the Bi-Encoder and Cross-Encoder are BERT-based models pretrained on English data, then fine-tuned on a subset of TR data.

The BERT-based retrieval process significantly outperformed a previous approach, which was based on BM25 [6], when retrieving similar TRs. Although model performance on identifying duplicates was also strong, two major drawbacks of the approach were identified:

1. The bi-encoder and cross-encoder fine-tuning process (i.e., how the models are trained), outlined briefly in Figure 1.3, does not effectively leverage the vast amount of general telecommunications language data present at Ericsson. Leveraging this domain-specific data before fine-tuning a model on the final downstream task has shown strong results on other tasks at Ericsson, as demonstrated in [2].

2. The models did not sufficiently generalize to TRs outside of the domain of TRs they were trained on. i.e., Poor performance on TRs written by different operators, which generally focus on different types of faults.

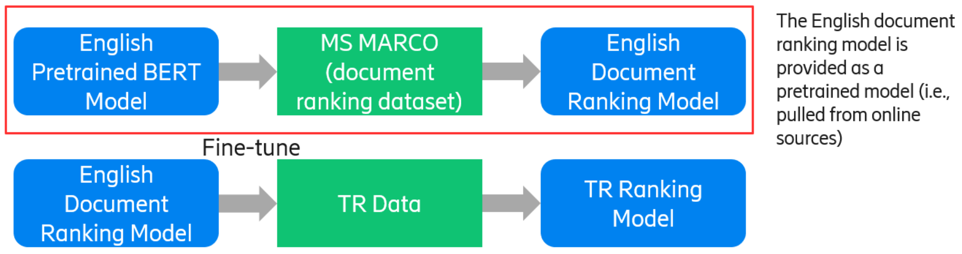

English Pretrained BERT Model
MS MARCO (document ranking dataset)
English Document Ranking Model
The English document ranking model is provided as a pretrained model (i.e., pulled from online sources)
Fine-tune
English Document Ranking Model
TR Data
TR Ranking Model

Figure 1.3: The model fine-tuning strategy in the original ranking approach outlined by [1]. Note that only the final fine-tuning process (i.e., fine-tuning on the TR data) is done at Ericsson. No other telecommunications data is leveraged in this fine-tuning strategy.

To address the drawbacks of previous work, the aim of this thesis is to identify and evaluate approaches to improve the performance of the models on duplicate TR retrieval by leveraging telecommunications-specific language data, focusing mainly on adapting the model fine-tuning processes. In addition to this, we focus solely on identifying duplicate TRs using our retrieval approach, which has implications for how the models are evaluated.

1.1.1 Prior Results

An initial attempt of integrating general telecommunications data into the model fine-tuning process was made at Ericsson in the summer of 2021. In this approach, the model is sequentially fine-tuned on general telecommunications language data (i.e., same type of text as the TRs, but for a more general task) and English document ranking data (i.e., different type of text to the TRs, but relevant for the final TR retrieval task). The aim of the fine-tuning approach was to encode the model with knowledge of both the relevant domain (telecommunications) and task (document retrieval) of the small available dataset of TRs before it has been fine-tuned on it. This is done under the assumption that the TR dataset is insufficient to train a strong model for TR duplicate retrieval, in the hope that providing the model with this additional, relevant data prior to final fine-tuning stage will lead to improved performance on duplicate TR retrieval..

Fine-tuning was applied sequentially in three stages:

1. Build a general telecommunications language model by fine-tuning a pretrained BERT model on telecommunications language data. This process is referred to as domain adaptation, where a model that has been trained to perform well on one domain (e.g., English language) is fine-tuned on a new domain (e.g., telecommunications language data). This process is outlined in further detail in Chapter 2.

2. Fine-tune the resultant model on English document ranking data, in the aim of *behaviorally* fine-tuning the model for the eventual downstream task.

3. Finally, fine-tune the model on a dataset of TRs. The resultant model from this stage can be used to retrieve duplicate TRs.

The aim of the above fine-tuning strategy was to minimally change the previous fine-tuning strategy outlined in Figure 1.3 as possible and can be seen in Figure 1.4. Note that the only change in this approach from the work presented in [1] is the addition of a fine-tuning step on telecommunications-specific language data.

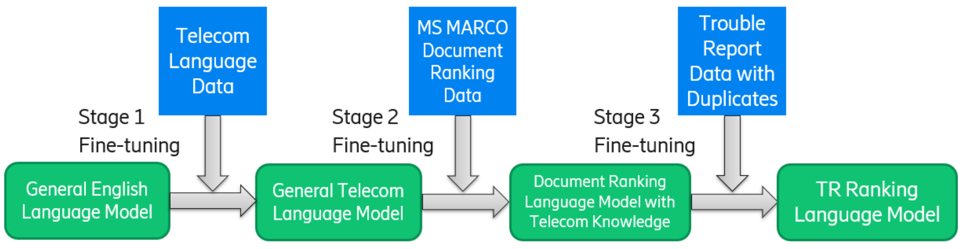

Telecom Language Data
Stage 1 Fine-tuning
MS MARCO Document Ranking Data
Stage 2 Fine-tuning
Trouble Report Data with Duplicates
Stage 3 Fine-tuning
General English Language Model
General Telecom Language Model
Document Ranking Language Model with Telecom Knowledge
TR Ranking Language Model

Figure 1.4: The fine-tuning approach in previous work. Note that we sequentially fine-tune the model on different datasets to best prepare it for the final downstream task.

This initial attempt found no significant difference in the final downstream model performance between models that had been fine-tuned on additional telecommunications language data and models that hadn't. At the time, the hypothesis as to why adding telecommunications knowledge to the model fine-tuning process did not leverage improved results centered around *catastrophic forgetting*. Catastrophic forgetting refers to the phenomenon where a model forgets previously learned information when optimizing for the new task [7, 8, 9]. Within the previously outlined fine-tuning process, after the model has been fine-tuned on general telecommunications data and is fine-tuned on an English document ranking dataset. In this second fine-tuning stage, there is no incentive for the model to retain the telecommunications knowledge that it had previously encoded.

From this prior work, two conclusions emerged: 1) further investigation into fine-tuning processes may be necessary to attain stronger performance on TR duplicate retrieval and 2) catastrophic forgetting may need to be mitigated for the fine-tuning process in Figure 1.4 to be effective. These conclusions are among the main points we wish to investigate in this thesis, as will be outlined further in section 1.1.2 and 1.2.

1.1.2 Research Questions

There are three central questions that we aim to explore in this thesis:

Transfer Learning/Fine-Tuning Strategies

First, **What transfer learning strategies for fine-tuning telecommunications-specific language models lead to the highest performance on a down-stream TR retrieval task?**

In a taxonomy outlined in [10], the two relevant fine-tuning strategies that could be applied in our scenario are **domain adaptation** and **sequential learning**. In **domain adaptation**, we take a model that has already been fine-tuned on the target task (i.e., document ranking/retrieval) and, with no structural changes to the model, fine-tune it on our target domain (e.g., fine-tune an English document ranking/retrieval model on our TR data, resulting in a TR/Telecommunications document ranking/retrieval model). Note that the fine-tuning approach outlined in Figure 1.3 is an example of domain adaptation. In **sequential learning**, we take a model that has been fine-tuned on the same domain as our target data, but needs to be fine-tuned to adapt to the task. For example, fine-tuning a general telecommunications model directly on the TR dataset is an example of sequential learning (domain remains within telecommunications, but the task shifts to document ranking/retrieval).

In addition to domain adaptation and sequential learning, we will also compare these strategies with the multi-stage fine-tuning strategy outlined in 1.1.1 to provide a full comparison of different transfer learning strategies. We hypothesize that a multi-stage fine-tuning strategy that does not suffer from catastrophic forgetting should be the most effective strategy, as it leverages both vast amounts of telecommunications data and document ranking data before the final fine-tuning on the TR data.

Mitigating Catastrophic Forgetting

Second, **How can we mitigate catastrophic forgetting and attain strong performance when we fine-tune a pretrained telecommunications-specific language model on a downstream TR retrieval task?**

Previous research indicates that methods such as Elastic Weight Consolidation (EWC) [11] consistently mitigate catastrophic forgetting, given that sufficient parameters are present in the model. Therefore, we hypothesize that the application of catastrophic forgetting mitigation strategies will 1) mitigate catastrophic forgetting and 2) improve our performance on our downstream duplicate TR retrieval task in our multi-stage fine-tuning framework as the model would retain relevant information from each fine-tuning stage.

Generalization to other TR Datasets

Third, and finally, **Which transfer learning strategies to fine-tune telecommunications-specific language models on a downstream TR retrieval task see the resultant models generalize well to TR datasets in different domains?**

We aim to evaluate our different transfer learning strategies on a TR dataset outside of the domain of TR data the models have been fine-tuned on (e.g., a dataset with different operators writing the TRs, on different types of faults).

1.2 Goals

To further expand on the previous section, there are three key objectives in this thesis:

1. We aim to compare domain adaptation, sequential learning, and our multi-stage (both with and without catastrophic mitigation strategies) fine-tuning strategies on the duplicate TR retrieval task. The aim of this

objective is to identify the most effective fine-tuning strategy for our task and potentially similar tasks in the future.

2. We aim to identify and evaluate catastrophic forgetting mitigation strategies, most notable EWC [11], in our multi-stage fine-tuning strategy for trouble report retrieval.

3. We aim to compare the performance of different fine-tuning strategies on an out-of-domain trouble report dataset, i.e., trouble reports that differ significantly from the reports the models were trained on. This provides insight into the generalizability of different approaches.

1.3 Sustainability and Ethics

There are no major ethical considerations that need to be made for this project. This thesis aims to aid the resolution of trouble reports and contribute to research on transfer learning strategies in natural language processing (NLP). For both these aims and the results we do not see any ethical concerns.

Regarding the impact of this thesis on sustainability, a notable concern is the energy consumption used in training large language models. Strubell, et al. (2019) [12] estimates that the CO_2 emissions of training a large transformer-based language model can exceed the average emissions of a car. In this thesis, we only fine-tune transformer models (i.e., starting with a pretrained model) as opposed to training the model from a random initialization. The emissions in this thesis should, therefore, not be as severe as other research in NLP. In addition to this, as the resultant work of this thesis could accelerate future fault resolution at Ericsson, this work may potentially displace and reduce other emissions in the fault resolution process.

1.4 Societal Impact

The results of this thesis are unlikely to have a major societal impact, as it is only relevant in fault resolution and NLP research. However, the potential deployment of the models outlined in this thesis would see interaction between the models and people trying to resolve TRs. This is an example of a human-in-the-loop model [13], in which a person uses the ML models as a tool, but all resolutions and conclusions are made through human intelligence and insight. Further innovation and creation of human-in-the-loop tools and models may have significant societal impact. With human intelligence at the center of the usage of ML, this can be seen as a fairer paradigm [13], indicating a positive societal impact.

1.5 Structure of the thesis

The remainder of this thesis is structured as follows. First, in Chapter 2, we discuss background topics relevant to the thesis, such as transformer networks, transfer learning, and catastrophic forgetting mitigation strategies. Second, in Chapter 3, we provide further details on the modelling task, the data, available models, and fine-tuning strategies. Third, in Chapter 4, we outline our experimental setup, evaluate model performance on the outlined experiments, and discuss the results in line with the goals outlined in 1.2. Finally, in Chapter 5, we conclude the thesis and outline directions for future work.

Chapter 2

Background

2.1 Overview

In this chapter, we provide an overview of the existing literature and background research relevant for this thesis. This is divided into three sections: 1) Transformer Networks & Neural Ranking, 2) Transfer Learning in NLP, and 3) Lifelong and Continuous Learning. We outline the focus/content of each section and provide insight as to how each of these sections are tied to the underlying goals of this thesis below.

Focus of Transformer Networks & Neural Ranking: The fundamental task that we aim to attain strong performance on is duplicate TR retrieval. As outlined in Chapter 1, there are two types of models used in this ranking process, notably bi-encoders and cross-encoders, each of which are BERT-based models. In this section, we therefore initially focus on outlining how transformer networks work and how they relate to BERT networks. Next, we outline how similar document retrieval can be performed using BERT networks, the different architectures available, and the re-ranking pipeline.

Focus of Transfer Learning in NLP: In this section, we outline how the different types of fine-tuning in NLP, we provide insight into how the parameters or BERT models change throughout fine-tuning, and outline some common challenges with this process.

Focus of Lifelong and Continuous Learning: In a multi-stage fine-tuning process, such as what was outlined in subsection 1.1.1, we often run into catastrophic forgetting, i.e., the model forgets information it learned previously in favor of effectively learning the current task. In this section, we provide a brief overview of lifelong and continuous learning and outline different forms of catastrophic forgetting mitigation strategies.

2.2 Transformer Networks & Neural Ranking

2.2.1 Transformer Networks

In recent years, the most used network architectures for language modelling are all based on the Transformer architecture outlined by Vaswami, et al. at NIPS 2017 [14]. Much like the earlier recurrent or convolutional sequence to sequence based models [15], the Transformer network consists of an encoder and decoder. An original sequence of text is fed to the encoder, which identifies meaningful representations of that text. The representations attained by the encoder are then fed to the decoder, which then generates text for the target task (e.g., translation). See Figure 2.1 for a simple example of what this process looks like.

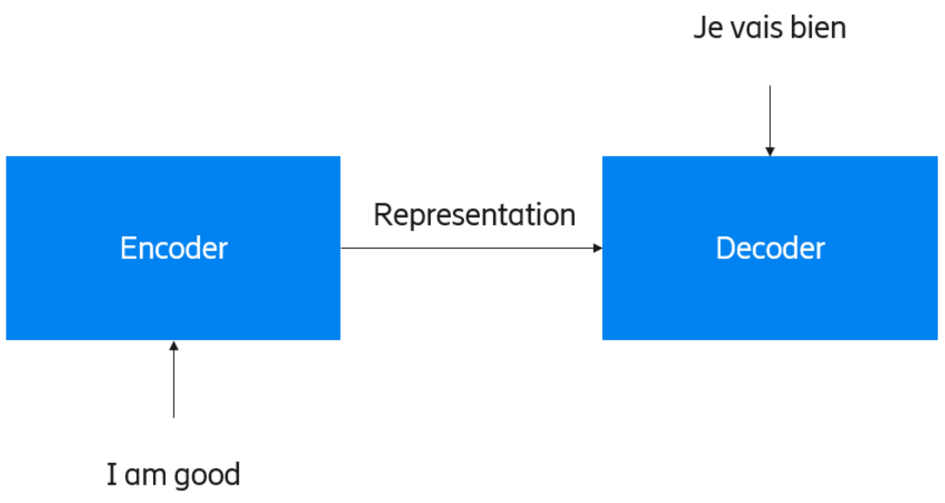


Figure 2.1: Simple structure of a transformer network. The encoder is provided a sequence of text and generates meaningful representations. The decoder takes those representations to generate some new text, in this case a French translation of the original input sequence.

Transformer Encoder

As outlined in [14], the encoder section of the Transformer consists of N encoder blocks, each of which is identical in structure. Each encoder block consists of a Multi-Head Attention layer and a standard feed forward layer, as is shown in Figure 2.2.

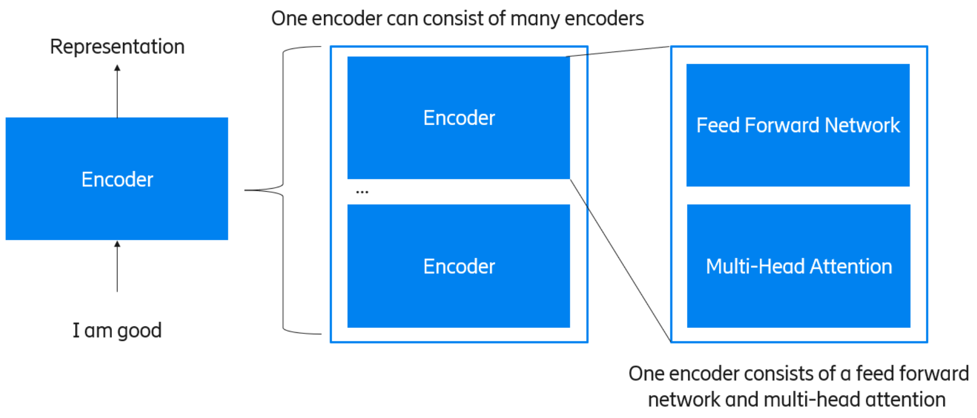

One encoder can consist of many encoders
Representation
Encoder
I am good
Encoder
...
Encoder
Feed Forward Network
Multi-Head Attention
One encoder consists of a feed forward network and multi-head attention

Figure 2.2: Visualization of the key elements in the Transformer Encoder.

Self-Attention and Multi-Head Attention The Multi-Head Attention block, introduced in [14], consists of multiple, parallel self (or scaled Dot-Product) Attention layers. This layer receives a matrix of non-contextualized token embeddings as input, e.g., given the sentence ”this is a sentence”, we have an embedding for each token [[0.991, 2.104, ..., 7.21], [...], [...], [...]] of which the size is the sentence length (4) times the embedding dimension (e.g., 512). Note that a token generally refers to a word or subword in the input text (e.g., the word ”run” would be considered its own token, but the word ”running” might be split into ”run” and ”ing”).

The key purpose of the self-attention layer is to contextualize the token embeddings [16]. In a good representation, we would expect the model to have some meaningful encoding of the impact that each token has on every other token (e.g., what impact does the word ”this” have on the word ”sentence” in the previous example?). This is done in the following process:

1. Given a matrix X, which consists of our original token embeddings, compute matrices Q (Query), K (Key), and V (Value). Where, $W^Q X = Q$, $W^K X = K$, $W^V X = V$. W is the weight matrix. The dimensions

of the Q, K, and V matrices are the sentence length (i.e., number of token embeddings in X) by some set dimension (e.g., 64). Note that, at this point, the query, key, and value matrices are simply a linear transform of the original input matrix X.

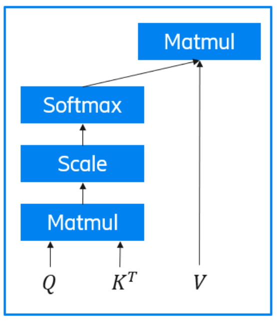
2. Compute QK^T, of which the result is a sentence length by sentence length size matrix. Intuitively, for each token (more specifically the token embedding) in the Query matrix, we compute the dot product between that word representation and all token embeddings present in the Key matrix. As the dot product provides us with a measure of similarity, every index position i, j within this matrix provides an indication of how similar the word at index i is to the word at index j.
3. Divide QK^T by the square root of the second dimension of the key vector (i.e., $\sqrt{d_k}$). This is used to achieve stable gradients.
4. Apply softmax to each row of QK^T. This is the only non-linear activation in the self-attention layer.
5. The final self-attention layer, Z, is attained by multiplying the resultant array from the previous step $softmax(\frac{QK^T}{\sqrt{d_k}})$ by the value matrix, V. Intuitively, this provides us with a weighted average of every word representation in V. The output matrix is the same size as V, i.e., sentence length by a set dimension (e.g., 64).

The entire attention process outlined above can be expressed in the following equation:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

We also outline the attention process in Figure 2.3.

One key drawback of the self-attention process is that the resultant representation matrix Z may only capture one element of the sentence structure or semantics. To account for this lack of capacity, the Multi-Head attention layer instead consists of h self-attention layers. For each self-attention layer, the matrices Q, K, and V are passed through linear layers to produce Q_i, K_i, and V_i. Then, we compute $Z_i = softmax(\frac{Q_i K_i^T}{\sqrt{d_k}})V_i$ for all i in $1..h$. The final representations $Z_{1..h}$ are concatenated and passed through another linear layer W_0.



Self-attention structure

Figure 2.3: Outline of the self-attention mechanism

Feed Forward Network The feed forward network in the transformer encoder is a simple 2 layer network, with ReLU activation after the first layer. i.e., $F(x) = ReLU(xW_1 + b_1)W_2 + b_2$. The ReLU activation function simply sets any negative values in $xW_1 + b_1$ to 0.

Add & Norm Although the key elements of each transformer encoder block are the Multi-Head Attention and the Feed Forward Network, between these layers and at the end of the encoder block there are Add & Normalization layers. In Figure 2.4, the full structure of a transformer encoder block is shown. The Add & Norm layers serve two purposes:

1. They serve as residual skip-connections, i.e., we *skip* a layer when propagating forward and backward. This can help mitigate exploding and vanishing gradients (e.g., [17]).
2. The normalization prevents large value changes, also mitigating exploding and vanishing gradients.

Input to the encoder The transformer model is provided a matrix X as input, where each row in X represents an embedding for an input token. These embeddings can be computed with a simple embedding matrix, which transforms a word token to a non-contextualized vector representation.

One key element of the input matrix X and the transformer model is that the positions of each token are not considered when computing the representations. This can be observed in the self-attention computation described previously. This is unlike previous approaches, e.g., recurrent neural network based sequence to sequence models where input is provided to

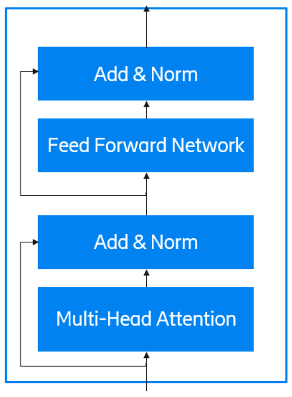


Figure 2.4: Full Transformer Encoder Structure.

the model in a sequential fashion. To provide the model with some knowledge of the position of tokens, an additional positional encoding is added to the model before providing the input to the transformer encoder. The positional encoding matrix, P, is the same size as the input data X, and is computed in the following way:

$$P(pos, 2i) = sin\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \tag{2.2}$$

and

$$P(pos, 2i+1) = cos\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \tag{2.3}$$

Where pos refers to the position of the token in the input sentence (i.e., the row in X), i refers to the dimension of the token embedding we are looking at, and d_{model} refers to the dimensionality of the model embedding.

When the positional encoding matrix, P, has been computed, it can be added to the input matrix X: $X + P$.

With all these steps in place, the final Transformer encoder can be seen in Figure 2.5.

Transformer Decoder

The structure of the entire transformer, including the decoder, can be found in Figure 2.6. Whereas the encoder generates a meaningful representation of an input sentence/document, the decoder generates the target text. For example, to translate an English sentence to French, the encoder will generate contextualized embeddings of the tokens in the sentence, whereas the decoder will use those embeddings to generate the equivalent sentence in French.

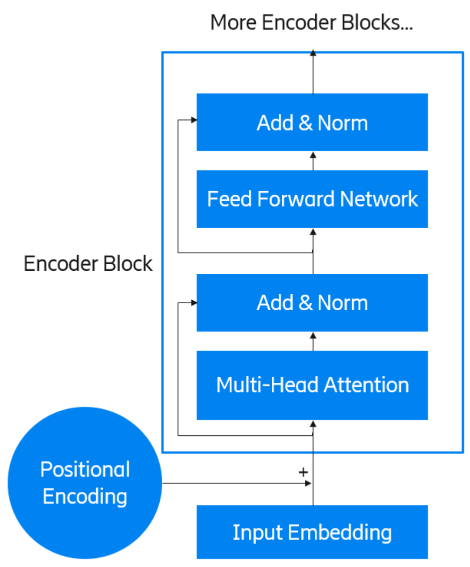


Figure 2.5: Full Transformer Encoder Structure

The decoder is used in the following way:

- Initially (at time=0), we provide the decoder with a token to indicate the start of a sentence (i.e., "<sos>"). The decoder then generates the next word in the sentence.
- At time=1, we provide the decoder with the start of the sentence token "<sos>" as well as the next token in the sentence (e.g., "Je"). Again, the decoder generates the next token in the sequence.
- This process continues until the decoder generates the end of sentence token ("<eos>").

Every decoder block is provided with the representation attained from the encoder in its Multi-Head Attention layer. Before this, however, note that the decoder consists of an additional type of block, referred to as a *Masked Multi-Head Attention* layer. We refer to [14] for a deeper overview of the Masked Multi-Head Attention layer.

BERT Models

The focus of this thesis is on BERT-based models, which only use the encoder section of the transformer model outlined previously. BERT was introduced in 2018 by Devlin, et al [5] as a simple and scalable approach to generating meaningful token embeddings.

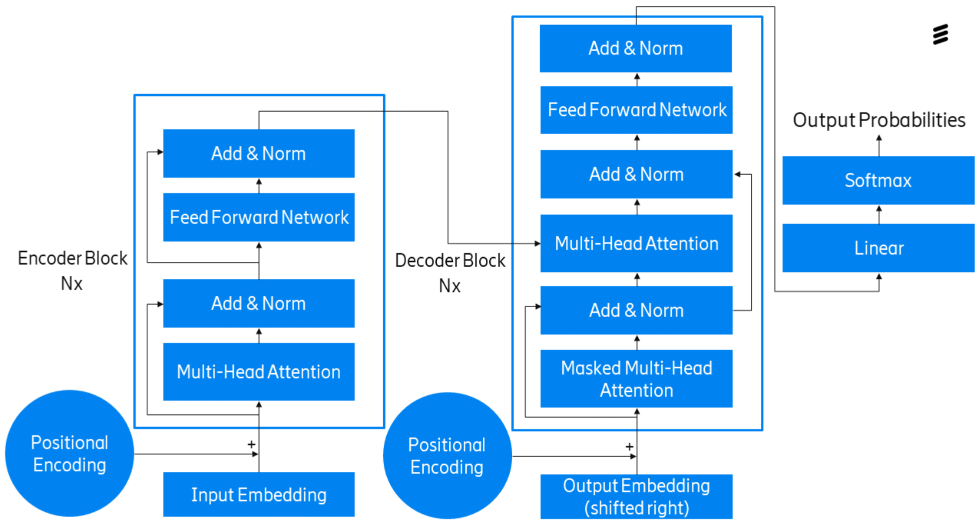


Figure 2.6: Full Transformer (Encoder and Decoder) Structure

Overview of BERT

The BERT model is a variation of the transformer encoder outlined in Figure 2.5. The base BERT model uses 12 encoder blocks, a hidden layer size of 768, and 12 attention heads per block. Rather than just using positional embeddings, BERT uses a sum of positional, segment, and token embeddings as input to the model.

The initial token embeddings are attained through the WordPiece tokenizer and embeddings [18].

BERT introduces a 2-stage framework for training large language models:

1. Pretrain a model on general language data (i.e., not for a specific task)
2. Fine-tune the model on a downstream task (e.g., Question-answering)

The pretraining is performed through two general tasks, namely Masked Language Modelling (MLM) and Next Sentence Prediction (NSP).

Training Details During pretraining, BERT is provided two concatenated sentences as input, with a separator token placed between them. The input to the transformer is the following, "[CLS] sentence_A_tokens [SEP] sentence_B_tokens [SEP]", where [CLS] is a special starting token and [SEP] is a separator token.

Of this input, 15% of the input tokens are randomly chosen for a masking procedure:

- 80% of the chosen tokens are replaced with a [MASK] token
- 10% of the chosen tokens are replaced with a random token
- 10% of the chosen tokens remain the same

In the MLM task, the model is trained to predict these true token for all of the randomly chosen tokens. This is trained through cross-entropy loss. i.e., Through an additional predictor layer, the network uses the contextualized embedding of a randomly chosen token to predict the true token. In this task, the model has to use the context in which the masked token exists to predict the token.

In addition to the MLM task, the model is also trained on NSP (next-sentence prediction). As outlined previously, the model is fed two sentences simultaneously. In 50% of cases, the second sentence follows from the first sentence, whereas otherwise it is a randomly selected sentence. In the NSP task, the token embedding attained from the [CLS] token is used to predict if the second sentence follows from the first. In this task, the model needs to attend to more tokens to be able to make an accurate judgement.

By training on these tasks, the BERT model outperformed other state-of-the-art language models at the time.

Note: As sentences will be of different length, inputs are padded to ensure that the length of each input will be the same. This is done using an attention mask where the value of every [PAD] token will be set to 0.

BERT Variations Since the release of the original BERT model, several variations have emerged. The following are some examples of these models and the differences between them:

1. **ALBERT:** 1) Uses cross-layer parameter sharing and 2) factorized embedding layer parameterization. [19]
2. **RoBERTa:** 1) Uses dynamic masking instead of static masking in the MLM task, 2) removes the NSP task, and 3) uses the BytePiece tokenizer over the WordPiece tokenizer. [20]
3. **ELECTRA:** Is trained in a generator-discriminator setup. The generator is fed a sentence with masked tokens and replaces them with the most likely tokens. The discriminator tries to identify the tokens the generator replaced. NSP is also removed. [21]

2.2.2 Neural Ranking with BERT

In recent years, Neural Ranking approaches, which refer to a set of neural network-based approaches to text ranking, have demonstrated significant performance increases over simpler exact-match retrieval approaches, such as BM-25 or TFIDF [6, 22]. The reason for this improvement in performance is the ability for deep neural networks to encode the context of tokens and sentences in its embeddings (i.e., a hidden layer representation of an input). An advantage of this is that the models are not reliant on an exact match of tokens. For example, in BM25 if two separate documents used slightly different terminology for similar concepts (e.g., using ’dog’ or ’hound’) then the computed similarity between those two documents may suffer as the model is unable to identify that the terms are related (i.e., no exact match between tokens). In a deep learning-based approach, the model may learn that the word ’dog’ and ’hound’ often appear in very similar contexts, allowing the model to more accurately identify similar documents [23].

In this section, we will initially introduce document ranking/retrieval and some of the key terminology used. After which, we will discuss pre-transformer neural ranking approaches. Finally, we discuss transformer-based neural ranking, including bi-encoders and cross-encoders.

Brief Introduction to Text Ranking

Text ranking is a common task in information retrieval. Generally, the goal of a text ranking approach is to rank a set of documents based on their relevance to a query document [24, 25]. For example, a search engine may compare a user query to millions of candidate documents (e.g., websites) and return the top documents ranked by their relevance to the query.

In text ranking, there exist three key elements, outlined both below and in Figure 2.7:

1. **Corpus:** A set of textual documents. For example, a large set of TRs.
2. **Query:** A textual document. For example, a newly written observation of a fault.
3. **Ranking Model:** A model that, given a query, returns an ordered list of documents from the corpus. Generally, the rank of a document in the corpus relates to the similarity between that document

In the models we discuss in this thesis, the ranking model follows a 2-stage procedure: 1) Produce a score for each document based on its relevance

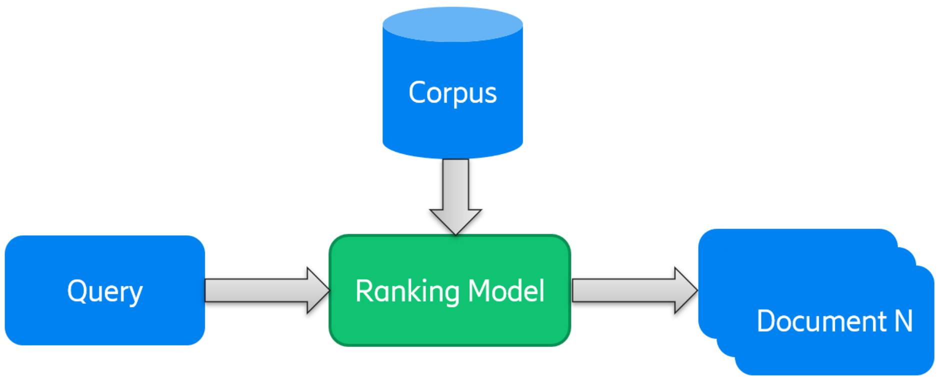


Figure 2.7: Overview of a Text Ranking Solution

to the query, where a higher score indicates higher relevance, and 2) Sort all documents based on their relevance scores.

Interaction and Representation-based Neural Ranking

Before transformer networks were introduced for neural ranking, most approaches can be classified as representation-based and interaction-based approaches [24, 26, 27, 28, 29, 30].

The **Representation-Based Approach** refers to approaches where the neural model produces an (often) high-dimensional embedding of an input text. Much like simpler approaches, similarity metrics like cosine-similarity or dot product can be used to compare these embeddings. Say we have a query document and a corpus of 1000 documents, we simply need to compute the embedding for the query document and all the corpus documents, and then check the similarity between the query and all corpus documents to rank their relevancy. A brief overview of how this comparison takes place is outlined in Figure 2.8.

The **Interaction-Based Approach** refers to approaches where the neural model receives two sentences or documents simultaneously and produces a score based on their similarity/relevance. In this approach, the neural model is often trained a as a ranking function, e.g., during training the model is provided pairs of documents that are tagged relevant or irrelevant, making it possible to train as a binary classifier. Interaction-based approaches often outperform representation-based approaches [24, 29, 30] as the model will be able to directly focus on the interaction between different tokens in the input query and corpus document. However, a caveat of this approach is that a separate forward

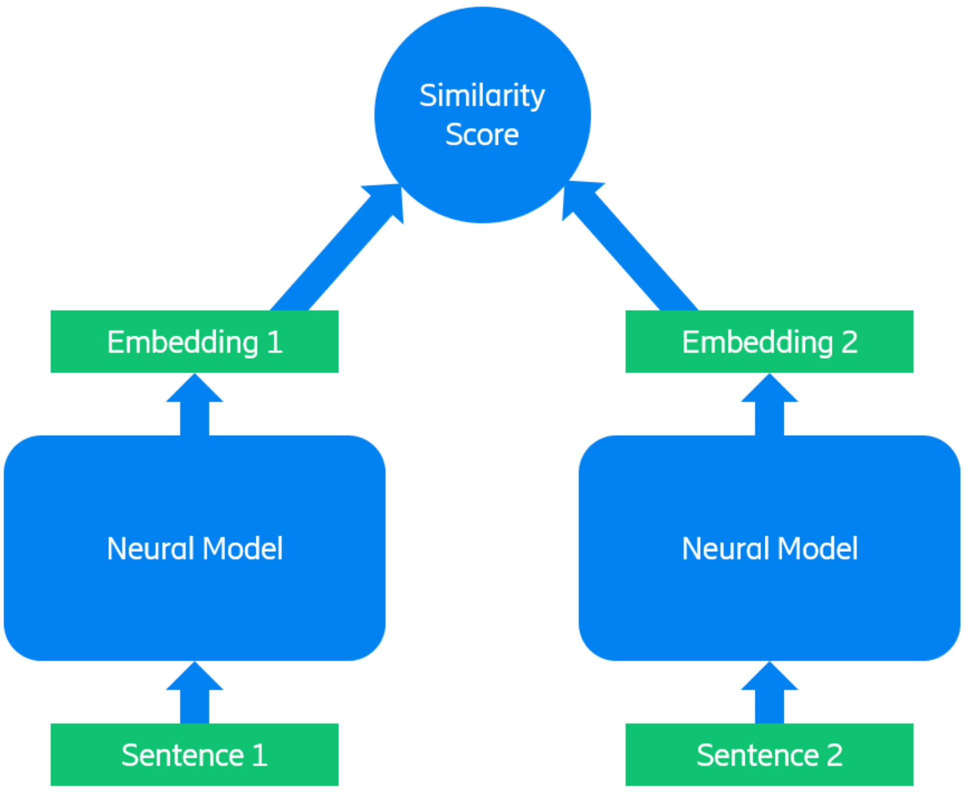


Figure 2.8: Overview of the Representation-Based approach for neural ranking.

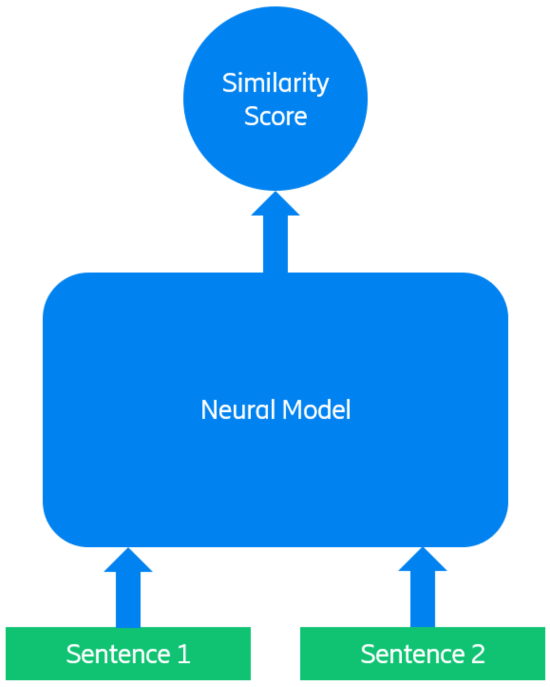


Figure 2.9: Overview of the Interaction-Based approach for neural ranking.

pass needs to take place for every comparison between documents. Rather than only performing one forward pass per document and then performing fast similarity computations on vectors, such as in the representation-based approach, the interaction-based approach will often be far slower. A brief overview of the interaction-based approach is outlined in Figure 2.9.

BERT-based approaches for neural ranking leverage many of the same concepts from pre-BERT neural ranking, but are instead grouped under bi-encoder and cross-encoder models.

Bi-Encoders

The Bi-Encoder is the BERT equivalent to a representation-based approach, in that it produces a representations of input documents which are then compared outside of the model (i.e., the model does not directly produce a score). One of the earliest Bi-Encoder models is *sentence-BERT* [31]. The sentence-BERT model looks very similar to Figure 2.8. By adding an additional pooling layer on the final token representations attained by a BERT model, it produces an embedding for an input sentence/document. Although these embeddings could immediately be used in a representation-based ranking setup, Reimers and Gurevych (2019) [31] find that additional fine-tuning of the model is beneficial.

Notably, by providing the model with examples of similar and dissimilar document pairs, the model can be fine-tuned in a Siamese network training structure [32], where the similarity score between similar documents is maximized and the similarity score for dissimilar documents is minimized.

There are other Bi-Encoder models which build on or are similar to the sentence-BERT approach, e.g., [33, 34, 35].

Cross-Encoders

Cross-Encoders can be said to be the BERT equivalent of interaction-based approaches, in that a single model is provided two documents simultaneously and produces a relevance score as output. These methods are provided query and document tokens and return a similarity score. One of the earliest and most significant cross-encoder architectures is the *monoBERT* model [36]. In the monoBERT model, the query and document tokens are provided to a BERT model by adding a separator token between them (i.e., ”[CLS], query tokens, [SEP], document tokens, [SEP]”). The model is then trained using cross-entropy loss to classify if a query and document are similar or not. Other than monoBERT, there are several other cross-encoder architectures, e.g., [37].

This approach can be quite slow, especially as the number of queries and documents increase. Therefore, monoBERT is often only used after an initial retrieval of relevant documents using a faster model. This process is referred to as re-ranking, and it outlined in the next subsection.

Re-Ranking

As outlined previously, cross-encoder models are often higher performing but slower models than bi-encoders. In addition to this, BM-25 based retrieval is generally even faster than bi-encoders. To leverage high performing models on a large set of queries, we can leverage multi-stage ranking approaches, where simpler, faster models retrieve an initial set of documents that can be re-ranked by a more complex model. Say, for example, that are trying to find relevant documents from our corpus of one hundred thousand documents. For a single query, it would be more effective to use a BM-25 based approach to retrieve the top 100 most relevant documents, and then use BERT-based approaches for re-ranking those documents.

This re-ranking process can be outlined in Figure 2.10.

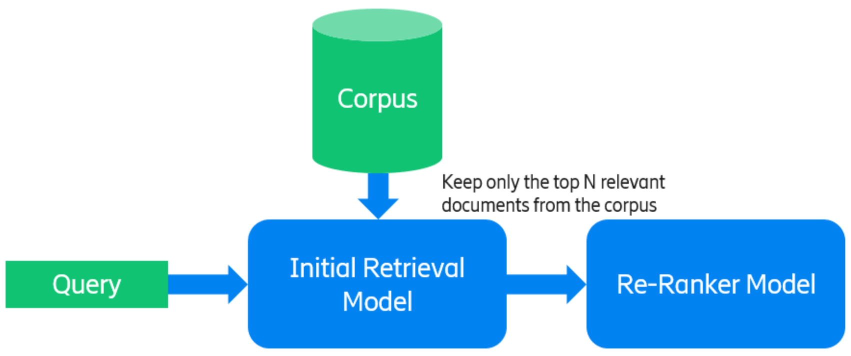


Figure 2.10: Re-Ranking process

Note that the initial ranker and re-ranker could both be BERT-based approaches. For example, using a bi-encoder approach as the initial retriever and the cross-encoder approach as the re-ranker has shown success [4]. In addition to this, multiple re-ranking stages could be applied, such as outlined by [36].

Evaluation Metrics

There are several metrics that can be used to evaluate the quality of a text ranking pipeline. We focus on two simple metrics to evaluate model quality: Recall@K and MRR@K:

- **Recall@K:** $\frac{Number\ of\ Relevant\ Documents\ in\ the\ Top\ K\ Retrieved\ Documents}{Number\ of\ Total\ Relevant\ Documents}$. e.g., if we have 20 total relevant documents for a given query and we see 10 in the top 10 retrieved documents, then our Recall@10 is 0.5. A high recall is a sign that the model is retrieving the necessary documents effectively, but it does not take the rank of documents into account.

- **MRR@K:** The reciprocal rank (RR) is the inverse of the rank of the highest ranked relevant document. For example, if we find that the highest ranked relevant document occurs at position 2, then our reciprocal rank is $\frac{1}{2} = 0.5$. Mean Reciprocal Rank refers to the mean of the reciprocal rank over many queries. MRR@K only computes the reciprocal rank for the top K documents for each query, i.e., if we only evaluate on a single query and the first relevant document occurs at position 4 then MRR@3 is 0, but MRR@5 is 0.25. A high MRR indicates that the model often ranks a relevant result among the top answers. It does not punish or reward the model if other relevant search results are high or low on the ranked list of documents, however.

2.3 Transfer Learning in NLP

As outlined in section 2.2, the purpose of the BERT training tasks is to produce contextualized token embeddings which can be used for downstream tasks. The direct application of these models to a new task or domain, however, may not lead to strong results. In the BERT framework [5], therefore, there are two stages for the application of a large language model:

1. *Pre-training:* Unlabelled model training on the base tasks.

2. *Fine-tuning:* Fine-tune all parameters using labelled data associated with the downstream task.

In this framework, a pre-trained model can be fine-tuned to different tasks and domains with minimal architectural changes.

2.3.1 Types of Fine-Tuning

In traditional machine learning, we generally assume that the domain and task a model was trained for remains static. By domain, we refer to the distribution of the data, e.g., although there is overlap, we consider general English a different domain of text compared to telecommunications or medical text data (different word usage, potentially a shift in grammatical structure). By task, we refer to the structure of model output and its optimization criterion throughout training, e.g., a classification model has very different output compared to a bi-encoder model. We often expect domain and task to remain static for the entire usage of a model. In deep learning applications, however, we often see a shift in both the domain and task on which the model is applied.

For fine-tuning and transfer learning, Pan, et al. [38] distinguishes between three transfer learning scenarios, notably inductive, unsupervised, and transductive transfer learning. As can be observed in Table 2.1, inductive transfer learning refers to when a model is fine-tuned on a new task, transductive transfer learning refers to when a model is fine-tuned on a new domain, and unsupervised transfer learning sees a shift in both task and domain. Fine-tuning a model trained on document classification on a document ranking task is an example of inductive transfer learning. On the other hand, fine-tuning a model that has been trained on only general

English language data to telecommunications language data is an example of transductive transfer learning.

Table 2.1: Types of Transfer Learning according to [38]

Type of Learning	Source and Target Domains	Source and Target Tasks
Inductive Transfer Learning	Same	Different
Unsupervised Transfer Learning	Different	Different
Transductive Transfer Learning	Different	Same

To account for recent advancements in NLP, Ruder (2019) [10] extends Pan's taxonomy for types of transfer learning in NLP. Notably, within transductive transfer learning Ruder distinguishes between domain adaptation, where the model adapts from one domain to another (e.g., general english to telecommunications language data), and cross-lingual learning, where a model is fine-tuned on another language entirely. In addition to this, in inductive transfer learning a distinction is set between multi-task learning, where a model is fine-tuned on multiple tasks simultaneously, and sequential learning, where a model is fine-tuned on multiple tasks sequentially.

Table 2.2: Types of Transfer Learning in NLP according to [10]

Class of Learning	Subclass	Description
Inductive Transfer Learning	Multi-Task Learning	Learn tasks simultaneously
Inductive Transfer Learning	Sequential Learning	Learn tasks in sequence
Transductive Transfer Learning	Domain Adaptation	Adapt model domain
Transductive Transfer Learning	Cross-Lingual Learning	Adapt model language

In practice, sequential learning is the most common type of transfer learning in NLP. This generally takes the form of training a model to a general task (i.e., the pretrained model) and the fine-tuning it on a new task, for example fine-tuning a pretained BERT model on a question answering task (same domain/type of text data, but different expected output). Sequential learning in scenarios where we aim to learn more than one task is also referred to as **lifelong learning** [39]. Lifelong learning will be discussed in further detail in section 2.4.

Types of Transfer Learning in this Thesis

Other than our multi-stage fine-tuning strategy, in this thesis we focus mostly on sequential learning and domain adaptation as transfer learning strategies.

In **domain adaptation**, we assume that our source task and our target task are the same, but that the domain is different. To fine-tune a model for this scenario, we can essentially just continue training the model as no structural changes need to take place [40].

In **sequential learning**, we assume that there is a difference between the source and target tasks. Because of this, there may need to be some structural change to the model (e.g., adding a pooling layer at the end for a bi-encoder, or a predictor layer for a classifier).

For both domain adaptation and sequential learning, choosing lower and dynamic learning rates have shown to improve model results [41, 42, 43].

2.3.2 What happens when we fine-tune BERT models?

BERT models encode key linguistic features in their representations [44, 45]. In addition to this, different layers (encoder blocks) in the BERT model may also contain different features. For example, word positional information is represented in lower layers in the network [46], syntactic information is often found to be most common in the middle layers of the network [44], and task-specific information is generally found in higher-level layers in the network [47].

When fine-tuning a BERT model to a new task, we would therefore expect that the higher level layers (i.e., the task-specific layers) change the most. [48, 49, 47, 50] show that this is most likely the case.

Wallat, et al. (2021) [51] found that knowledge in the network (e.g., factual information present in the training data) is somewhat dispersed through both the intermediate and final layers of the model. When fine-tuning models to different tasks and data, a loss in previously known facts was observed. However the severity of this loss was highly dependent on the type of fine-tuning task, with retrieval and ranking tasks leading to a model retaining more information than when fine-tuning on question answering tasks. The results of this are also highly dependent on the data the model is trained on.

2.3.3 Common Challenges

When fine-tuning models to smaller, downstream domains and tasks we often run into several challenges. Notably, overfitting on the new data is possible, leading to poor generalizability. In addition to this, catastrophic forgetting is a well known issue in fine-tuning language models [42].

Brief Introduction to Catastrophic Forgetting

When fine-tuning a model on a new dataset, it is possible that the parameters which allowed the model to attain high performance on the previous task change significantly to attain high performance on the new task. The reason for this is simple: the performance on the previous task is no longer important, hence all parameters should be updated to optimize for the new task. In practice, this may not represent a major issue as we often only care about the performance on the final task. However, it is a major challenge for building generalizable models that can perform well across multiple tasks [10, 52].

In the next section, we will further outline the challenges associated with catastrophic forgetting and several mitigation strategies.

2.4 Lifelong and Continuous Learning

Continual and lifelong learning refers to the scenario where we expect a model to learn new tasks and environments in a continuous fashion, while avoiding catastrophically forgetting previously learned information [53]. Although, similarly to other transfer learning scenarios, we aim to build a model that performs well on a duplicate TR retrieval task, we can still leverage lifelong learning research to aid our multi-stage fine-tuning process. In the multi-stage fine-tuning strategy outlined in section 1.1.1, we aim to build a model that sequentially learns both telecommunications-relevant and document-ranking knowledge without suffering from catastrophic forgetting. In this section, we will discuss catastrophic forgetting in more detail, focusing strongly on potential mitigation strategies.

2.4.1 Catastrophic Forgetting

When deep learning models are trained on new tasks or domains sequentially, the models have a tendency to forget information related to earlier tasks in favor of learning new information. This phenomena is referred to as catastrophic forgetting, which represents a major challenge in sequential and lifelong learning [11].

Although catastrophic forgetting has been observed in neural networks for decades [7, 54], much recent research aims to address it to counter its consequences, especially with recent advances in transfer learning and reinforcement learning where sequential learning of multiple tasks becomes necessary [11].

In Natural Language Processing, catastrophic forgetting represents a major challenge in effectively fine-tuning large, language models [43, 42, 10]. It is undesirable for a model to lose knowledge of previous domains when fine-tuning on a new domain. For example, a model trained on standard English through masked language modelling should not lose that knowledge when fine-tuning on a smaller subset of medical record data.

2.4.2 Metrics for Measuring Catastrophic Forgetting

For a set of tasks T that a model needs to learn sequentially, we construct a matrix R which has dimensionality $T \times T$. Upon training on task 1, t_1, we attain performance (e.g., accuracy) $r_{1,1}$. When we fine-tune on task 2, not only do we observe $r_{2,2}$, but we can also observe the accuracy of the model on the previous task $r_{2,1}$. If we observe a significant decrease in performance from $r_{1,1}$ to $r_{2,1}$ then we can say that the backwards transferability of the model is poor (i.e., catastrophic forgetting is occurring).

Each index pair i, j in R contains the performance (commonly accuracy) that the model attains on task t_j after having been fine-tuned on task t_i. Note that it is possible to evaluate on, for example, task 2 while the model has only been fine-tuned on task 1.

Given this setup, Lopez-paz, et al (2017) [55] defined three fundamental metrics for continual learning:

1. **Average Accuracy (A):** $\frac{1}{T}\sum_{i=1}^{T} R_{T,i}$, which represents the average accuracy across all tasks after having sequentially trained on all tasks
2. **Backwards Transfer (BWT):** $\frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i}$, which represents the average difference in model performance on each task after having

trained on all tasks sequentially compared to when it has most recently fine-tuned on that task.

3. **Forward Transfer (FWT):** $\frac{1}{T-1}\sum_{i=2}^{T} R_{i-1,i} - \bar{b_i}$, which represents the performance of a model on a task it hasn't trained on yet, compared to the performance of a randomly initialized model (i.e., how much did training on task 1 help my performance on task 2?). This is especially useful in a zero-shot learning context.

Díaz-Rodríguez, et al. (2018) [56] proposed additional metrics to better measure continual learning, such as measuring the model size, storage, and computational efficiency of models throughout fine-tuning. In addition to this, the authors separate the BWT metric into two separate metrics: 1) *REM*, or remembering, is defined as $1 - |min(BWT, 0)|$ and 2) *BWT+*, or positive backwards transfer, is defined as $max(BWT, 0)$. These metrics aim to explicitly measure how much the model retains when training on new tasks (through REM) and if there is any improvement on previous tasks after having trained (BWT+).

2.4.3 Catastrophic Forgetting Mitigation Strategies

There are many strategies for mitigating catastrophic forgetting in deep learning models, both in the general case and more specifically for language models. The aim of these methods is to find a parameterization of a model which can attain low loss on multiple tasks. We can interpret catastrophic forgetting as the scenario outlined in Figure 2.11. In standard fine-tuning, a model greedily shifts from a parameterization which achieves low error on task 1 to a parameterization that achieves low error on task 2 [11].

According to Zenke, Poole, and Ganguli (2017) [57], catastrophic forgetting mitigation strategies can be categorized as:

1. **Architectural:** Alter the structure of the network without altering the objective function. Freezing network weights is an example of this, or parameter/layer-wise learning rate reduction.

2. **Functional:** Adds a regularization term to the objective function, with the aim of preserving the input-output mapping of the old task. This is often computationally expensive, as it can require using the old network to compute network output. An example of where this is used is in knowledge distillation.

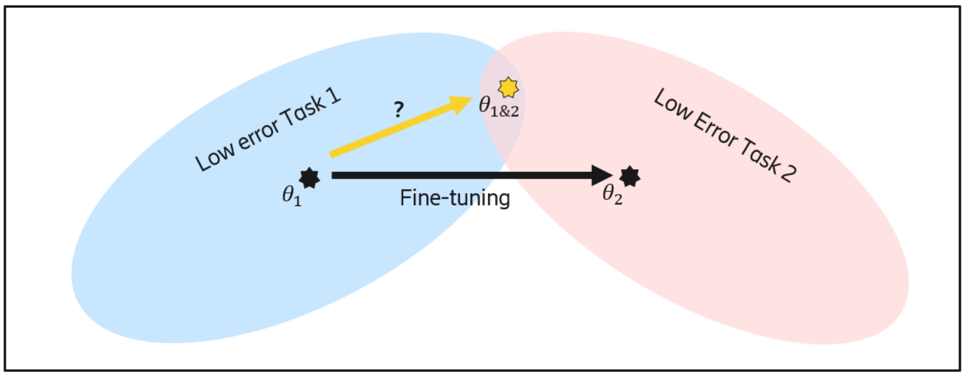


Figure 2.11: Outline of what often happens during fine-tuning, where we leave the model parameterization which achieves low error on an initial, source task to achieve low error on the target task. Ideally, with catastrophic forgetting mitigation strategies, we would aim to achieve low error on both tasks simultaneously.

3. **Structural regularization:** Regularizes parameters such that sets of them remain similar to the original task. An example of this is elastic weight consolidation [11].

Elastic Weight Consolidation

Kirkpatrick, et al. (2017) [11] proposed elastic weight consolidation (EWC) as a way to mitigate catastrophic forgetting when learning multiple tasks sequentially. By adding a quadratic regularization penalty, EWC constrains the parameters which were important for an initial task (e.g., task A) when learning on a new task (e.g., task B).

EWC uses the Fisher information matrix [58], F, to estimate the importance of each parameter in the model. As can be seen in the equation below, the constraint on each parameter is based on the squared difference between the parameter value on task A and its current value, multiplied by the importance of that parameter $F_i(\theta_i - \theta^*_{A,i})^2$.

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2 \tag{2.4}$$

The loss, $\mathcal{L}$, for the model is as described above. The additional regularization will punish the model for significantly changing important parameters. A practical note is that the Fisher information matrix can be approximated by the squared first-order gradients in the model [58].

EWC has been applied to deep language models previously, e.g., [59, 52, 60]. Although the results of this research does indicate that catastrophic forgetting is mitigated, it does not demonstrate the level of improvement seen in the continuous learning literature.

Gradient Episodic Memory

In Gradient Episodic Memory (GEM) [55], a memory $\mathcal{M}_t$ of some observed examples of a previous task t is stored. In addition to computing the loss on the existing task g_t, the loss is also computed for all previous tasks:

$$l(f_\theta, \mathcal{M}_k) = \frac{1}{|\mathcal{M}_k|} \sum_{(x_i,k,y_i)\in\mathcal{M}_k} l(f_\theta(x_i, k), y_i) \tag{2.5}$$

Where k is the task, x_i and y_i are samples and labels from the dataset associated with task k, and f_θ is the current model. The total loss for GEM is slightly more complicated than the approach outlined above. We refer to [55] for a more detailed discussion.

Unlike EWC, GEM can encourage positive BWT. However, it requires a memory of some previously seen datapoints, which is not always feasible.

Synaptic Intelligence

In the Synaptic Framework outlined by Zenke, Poole, and Ganguli (2017) [57], each weight (synapse) is assigned a measure of importance. Much like in EWC, Synaptic Intelligence (SI) introduces an additional term to the loss function of a network when training on a new task:

$$\tilde{\mathcal{L}_\mu} = \mathcal{L}_\mu + c\sum_k \Omega_k^\mu(\tilde{\theta_k} - \theta_k)^2 \tag{2.6}$$

Where $\mathcal{L}_\mu$ refers to the loss on the original task, c is a tunable hyperparameter, $(\tilde{\theta_k} - \theta_k)^2$ refers to the difference between the parameterization of the model on the previous task and the current parameters, and Ω_k^μ refers to the regularization strength of each parameter. Ω_k^μ can be further defined as:

$$\Omega_k^\mu = \sum_{(v<\mu)} \frac{w_k^v}{(\Delta_k^v)^2 + \xi} \tag{2.7}$$

Where ξ is a damping parameter, Δ_k^v is the difference between the parameter k on the task v and task $v-1$, and w_k^v refers to the parameter k's contribution to the loss on task v. Note that Ω_k^μ is summed for all tasks up to the current task μ.

Synaptic intelligence is conceptually similar to EWC, but it often outperforms EWC due to the changes in its loss formulation. In this thesis, EWC is used as it is a fundamental approach in mitigating catastrophic forgetting and is simple to implement and understand. In future work, it would be worthwhile to experiment with multiple mitigation strategies.

ULMFiT

ULMFiT is an approach for fine-tuning language models outlined by Howard and Ruder (2018) [43]. Although it does not directly target mitigating catastrophic forgetting as previous approaches do, it outlines several improvements to fine-tuning by using triangular learning rates and gradual unfreezing of layers throughout fine-tuning. Although this research was not performed on Transformer-based models, more recent research also indicates that similar conclusions hold [42].

Chapter 3

Methods

In this chapter, we will provide an overview of the data, outlining basic dataset metrics, the structure of TRs, a brief exploration of our dataset, and the preprocessing details involved. After this, we provide an overview of our ranking pipeline, including details on initial retrieval and re-ranking. Finally, we discuss the fine-tuning process of our models, outlining the base models we have available and details regarding our implementation of all of the fine-tuning strategies.

3.1 Data Overview

3.1.1 Trouble Report Dataset

A trouble report (TR) is either a primary TR, i.e., ideally the first instance of the fault, or a duplicate TR, meaning that there exists some associated primary TR that described the fault first. Within our dataset of **28.5K** datapoints, we find **23K** primary trouble reports and **5.5K** duplicate trouble reports. The small amount of duplicate trouble reports indicates that directly training a model to classify if a report is a duplicate or not may not be possible.

3.1.2 TR Structure

Within a trouble report there are several key sections that describe the identified fault. As shown in Figure 3.1, we look into four sections:

- **Fault Tag:** Outlines in what general region the fault occurred, e.g., it is found to be a radio software issue.

- **Header:** Outlines an initial and short description of the fault
- **Observation:** A much longer section that describes the fault in detail. This often includes log output.
- **Answer:** After the trouble report has been answered, the answer section will be filled with a description of what went wrong and the solution or next steps.

In this thesis, we concatenate the Fault Tag, Header, and Observation to construct our final observation, as this contains all meaningful observed data.

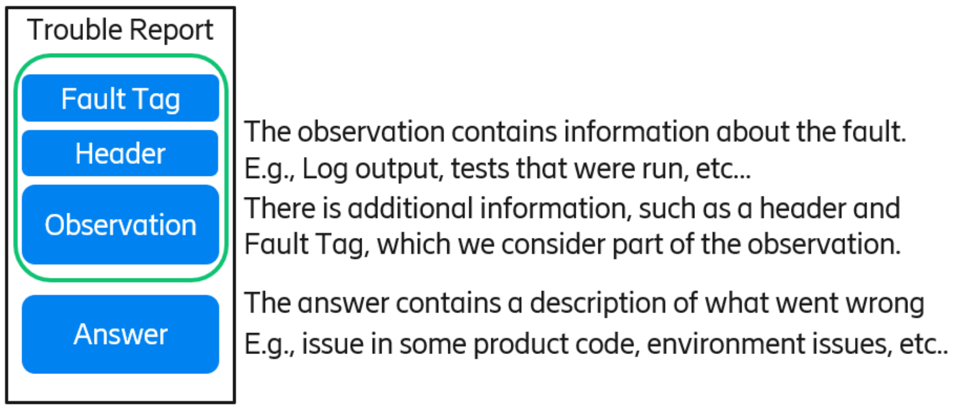

Trouble Report
Fault Tag
Header
Observation
Answer
The observation contains information about the fault.
E.g., Log output, tests that were run, etc...
There is additional information, such as a header and
Fault Tag, which we consider part of the observation.
The answer contains a description of what went wrong
E.g., issue in some product code, environment issues, etc..

Figure 3.1: Overview of the structure of a TR. The key elements are the observation and answer sections. We consider the fault tag and header to be part of the observation section.

3.1.3 Data Exploration

Our dataset focuses on a subset of trouble reports collected over the last three years. Out of the 28667 datapoints, we see a fairly even distribution over trouble reports over the three years, as outlined in 3.2.

A key challenge when we observe our data, however, is that the length of the observation and answer section may exceed the limit of tokens that our models can take as input. In Figures 3.3 and 3.4, we show the number of words in each TR's observation and answer sections, respectively. We find an average of approximately 700 words in the observation section and 200 in the answer section. Although we may expect the answer section to remain within

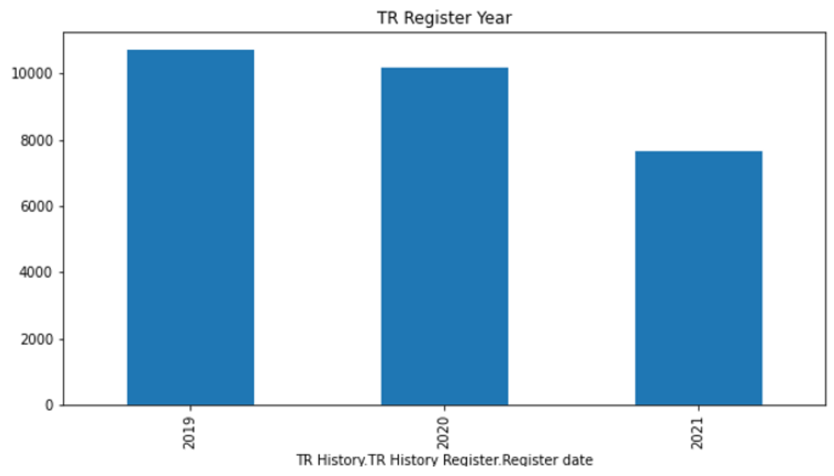


Figure 3.2: Number of TRs registered per year in our dataset

the necessary size for our models, the observation section will often need to be truncated. This runs the risk of losing vital information in our observation section, which may reduce model performance.

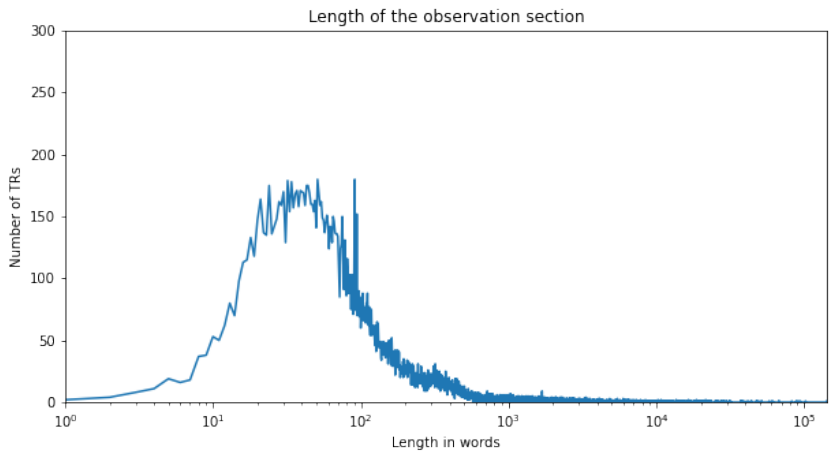


Figure 3.3: Number of words in the TR observation sections

3.1.4 Preprocessing Details

We aim to provide the model with as unprocessed data as possible, as we found significant preprocessing (e.g., removing numbers and stop words) reduced the model performance. The preprocessing we did perform was the following

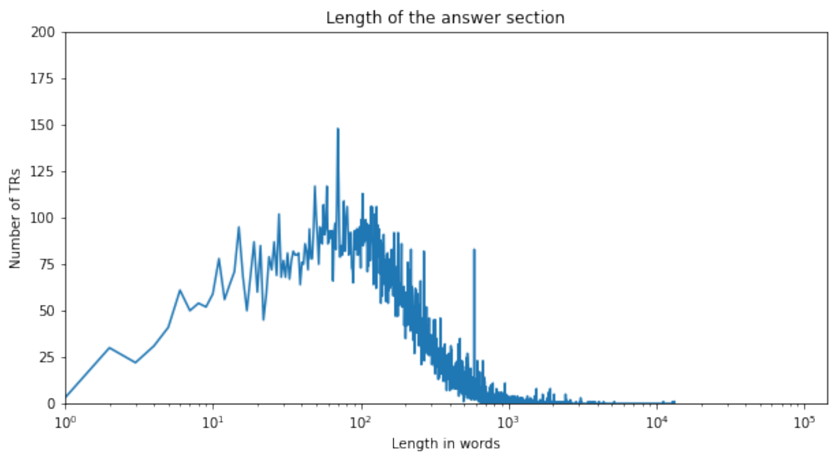


Figure 3.4: Number of words in the TR answer sections

(note that this process is performed separately for the observation and answer sections):

1. Extract certain relevant sections of text to reduce the data size. This selection was made based on an existing TR parser.
2. Concatenate the strings of relevant sections from the previous stage.
3. Remove repeating whitespace and newlines.
4. Tokenize the dataset through Spacy [*], a powerful NLP Python library
5. For each token, identify if it matches a known abbreviation (e.g., 5G) and replace it with the original terms.
6. Re-concatenate all tokens to create the final strings.

Due to some datapoints with missing values or too short observations/answers, the total dataset size is reduced from 28.5K to 24.5K TRs, of which 3.5K are duplicate TRs.

3.2 TR Ranking Overview

As outlined in chapter 1, previous work [4] outlined a 2-stage ranking system for duplicate TR retrieval, shown in Figure 1.2. This ranking approach uses

[*] https://spacy.io/

two BERT-based retrieval models: 1) A bi-encoder that efficiently retrieves the top 20 most relevant documents and 2) a cross-encoder that re-ranks the provided 20 documents.

3.2.1 Initial Retrieval with the Bi-Encoder

The initial retrieval process with the bi-encoder model can be seen in Figure 3.5. The bi-encoder model produces an embedding for the TR observation and all texts (TRs) in the corpus. After the embeddings have been produced, a nearest neighbor search is run using cosine similarity as a distance metric to find the 20 closest TRs to the TR observation. We save the IDs of these TRs for the re-ranker stage.

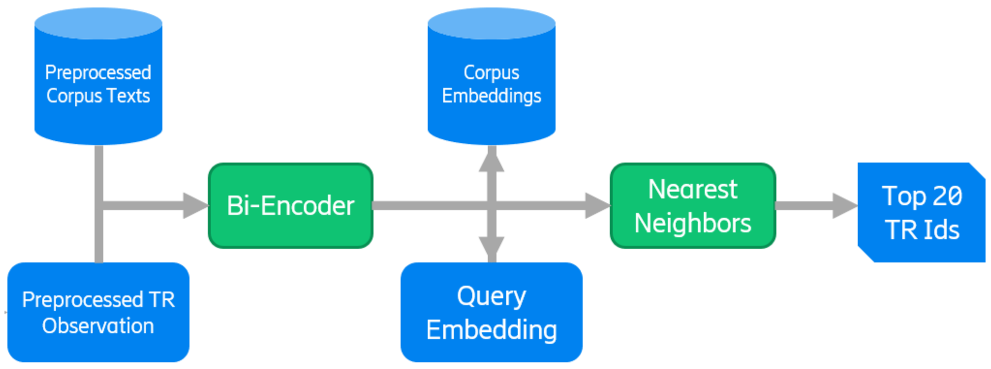


Figure 3.5: Initial Retrieval with the Bi-Encoder

3.2.2 Re-Ranking with the Cross-Encoder

The re-ranking process is outlined in Figure 3.6. Given the top 20 TRs from the initial retrieval, the cross-encoder is used to compute a similarity score between each TR and the TR observation query. Finally, each TR is then ranked based on its similarity to the TR observation.

3.3 Model Fine-Tuning Process

3.3.1 Available Models

Several transfer learning strategies with different models are outlined in throughout the remainder of this thesis. An overview of the models mentioned are the following:

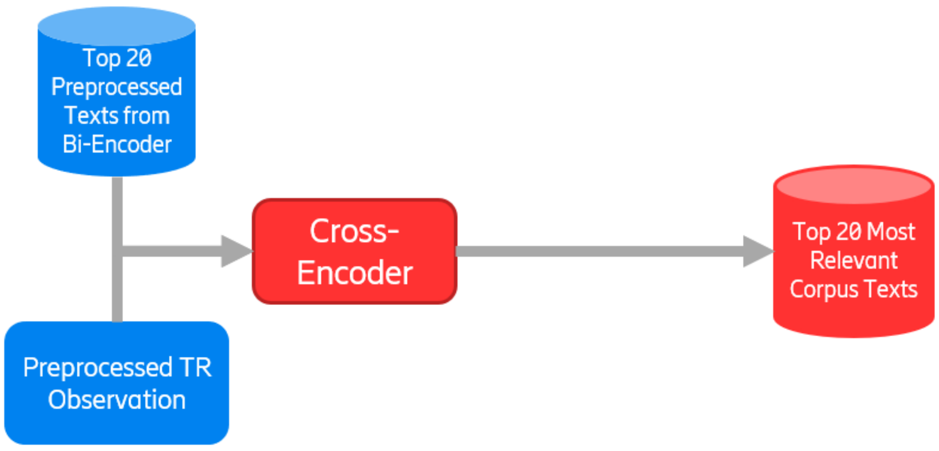

Top 20 Preprocessed Texts from Bi-Encoder
Preprocessed TR Observation
Cross-Encoder
Top 20 Most Relevant Corpus Texts

Figure 3.6: Re-ranking with the Cross-Encoder

- **RoBERTa:** [20], a model trained on 160 GB of general English data through dynamic masked language modelling. This is the base model for TeleRoBERTa and the MS MARCO models.
- **TeleRoBERTa:** [2], the same as the RoBERTa model, but with continued pretraining on 21 GB of general telecommunications data through dynamic masked language modelling. The telecommunications data includes TR and open 3GPP specifications data [*]. 3GPP specifications contain specifications for global mobile broadband standards.
- **MS MARCO model:** we have bi-encoder and cross-encoder versions of MS MARCO model, both of which are based on the RoBERTa model. The details for how these models were trained will be discussed further in section 4.1. MS MARCO [61] is a dataset consisting of hundreds of thousands of question answer pairs collected from search engines. It is often used to train question-answering and document ranking models.

3.3.2 Fine-Tuning Strategies

As outlined in chapter 2, there are several approaches in the existing transfer learning taxonomy which are relevant for our scenario. 1) **Domain Adaptation** consists of fine-tuning a model that has been trained on our target task but not on our target domain, which in our case is the MS MARCO model, on our TR data. 2) **Sequential Learning** consists of a fine-tuning a

[*] https://www.3gpp.org/specifications

model that has been trained on our target domain but not on our target task, e.g., TeleRoBERTa, on our TR data. These approaches are further outline in Figures 3.7a and 3.7b.

A key challenge with domain adaption and sequential learning is that we are unable to leverage all of available data for our task. Notably, it would be ideal if a model could be fine-tuned on both telecommunications data and MS MARCO data to ensure that it has been exposed to both the target domain and task before it is applied on the TR data. For this reason, we investigate a multi-stage fine-tuning approach, outlined in Figure 3.7c.

Note that in this approach we may observe catastrophic forgetting in the initial fine-tuning stage, where the MS MARCO fine-tuning stage may cause the model to forget the relevant telecommunications-specific knowledge. For this reason, in the multi-stage fine-tuning approach we investigate both the naive case and with a catastrophic forgetting mitigation strategy, notably using Elastic Weight Consolidation when fine-tuning on MS MARCO data.

The purpose of using EWC in the initial fine-tuning stage is to build a model that has both encoded the task and domain-specific information, such that this knowledge can be leveraged to attain higher performance when fine-tuning on the TR dataset.

In total, we train a bi-encoder and a cross-encoder model for each of the four scenarios:

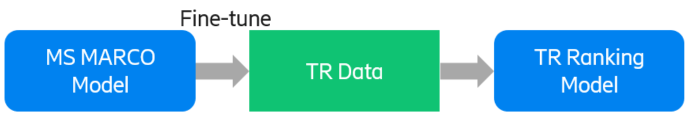
- **Domain Adaptation (DA):** Fine-tuning an MS MARCO model on the TR data.
- **Sequential Learning (SL):** Fine-tuning TeleRoBERTa on the TR data.
- **Multi-Stage Fine-Tuning (MS):** Fine-tuning TeleRoBERTa on MS MARCO, then on the TR data.
- **Multi-Stage Fine-Tuning with EWC (MS w/ EWC):** Fine-tuning TeleRoBERTa on MS MARCO with EWC, then on the TR data.

Elastic Weight Consolidation Implementation

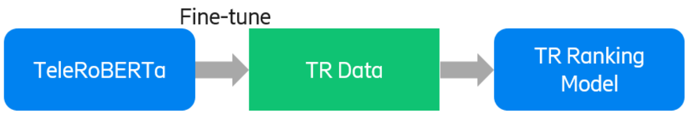
As outlined in 2.4.3, EWC requires us to compute a Fisher Information Matrix, which stores information about the importance of each parameter. We compute this in accordance with [58] by collecting the mean squared first order gradients of TeleRoBERTa when we run and evaluate on 7000 randomly selected lines of 3GPP specifications. Note that the gradients are collected when performing the dynamic masked language modelling task.

The selected 3GPP lines are part of the training data for TeleRoBERTa. An example of what a sentence looks like is the following:

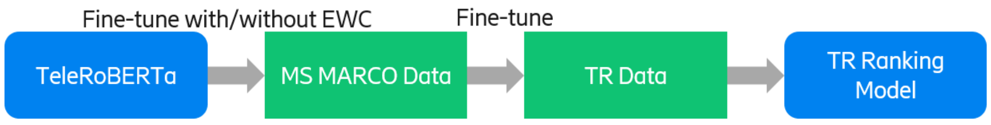
```
The IMS multimedia Telepony communication service
consists of two principal parts: a basic communication
part, and an optional supplementary services part.
```



(a) Domain Adaptation



(b) Sequential Learning



(c) Our multi-stage fine-tuning approach

Figure 3.7: The three possible fine-tuning approaches.

3.3.3 Fine-tuning on TR Data

Although the fine-tuning process on MS MARCO is simple due to the query, document pairs that is provided in the dataset, there are several potential strategies for fine-tuning the ranking models on the TR data. There are three simple approaches we identified that could be used to fine-tune our models to our duplicate retrieval task:

1. **Strategy 1: Primary and Duplicate Observation Comparison:** The most straightforward approach would be to fine-tune our models to produce high similarity scores when comparing a duplicate TR observation to a primary TR observation. This allows us to directly replicate the final task while fine-tuning. However, as the number of duplicates in our dataset is very limited, this approach may suffer from a lack of data both in the fine-tuning and evaluation stage.

2. **Strategy 2: Primary Observation and Primary Answer Comparison:** To leverage the vast quantity of primary TRs present in our dataset, another approach is to fine-tune the models to produce high similarity scores when provided a primary observation and answer. i.e., the model receives an observation and answer corresponding to the same TR and is fine-tuned to return a high similarity score. To apply our task to the TR duplicate retrieval task, a new duplicate observation is compared to all primary TR answers. With this approach, the model can be fine-tuned using all primary TRs. Challenges may emerge if the transferability to duplicate observations is low.

3. **Strategy 3: Hybrid Approach**: In previous work [4], strategy 2 was used. However, to ensure some transferability to duplicate TRs, a small subset of duplicate TR observations were added in the fine-tuning process (i.e., duplicate observation, primary answer pairs).

Although the 3rd fine-tuning strategy may ensure more transferability to the target task, to ensure the maximal amount of duplicate TRs would be available for evaluation we opted for Strategy 2.

Chapter 4

Evaluation

4.1 Experimental Setup

4.1.1 Dataset Split

Our data split strategy is straightforward. Initially, we identify all primary TRs which have no duplicates present in our dataset (i.e., they could not be used for evaluation). We construct our training dataset from the *observation* and *answer* sections of these primary TRs. The remaining primary TRs we split equally into validation and testing sets. For the validation and testing sets, we construct our *corpus* from the primary TR answers. As each of these TRs have some associated duplicate TRs, we construct our set of *queries* from the duplicate TR observations. The ranking task is evaluated based on if the model can identify the primary TR answer given a duplicate TR observation. The split metrics can be found in Table 4.1.

Table 4.1: Dataset split sizes

Dataset	Number of Duplicate TRs	Number of Primary TRs	Total
Train	0	18.5K	18.5K
Validation	1.2K	1.2K	2.4K
Test	2.5K	1.2K	3.7K

4.1.2 Different Evaluation Scenarios

Scenario 1: To evaluate our model's performance at identifying a duplicate TR, we construct the following evaluation scenario: *Given a duplicate TR observation, how well can our model retrieve the primary TR answer?*

Scenario 2: Our first scenario evaluation deviates slightly from previous work by [4], however, where the model was evaluated when using both duplicate and primary TR observations. To ensure parity with previous work, we investigate this scenario as well.

Scenario 3: The reason we shifted from scenario 2 to scenario 1 in this thesis is due to it better modelling the real-world situation. In addition to this, we hypothesized that the model would naturally perform better when retrieving a primary TR answer using the primary TR observation, as opposed to using a duplicate TR observation. We hypothesized this to be the case due to there potentially being numerous artefacts (e.g., logs, similar word usage, etc...) the are shared between the answer and observation of the same TR, hence it would not be a good real-world measure of the model performance. To identify if we can be confident in our hypothesis, we also investigate the scenario where we use only the primary TR observations to retrieve the primary TR answers.

Scenario 4: Finally, we also hypothesize that duplicate and primary observations share more information than a duplicate observation and primary answer would. Although the models will not be trained on identifying similar observations, we will evaluate the models at retrieving the correct TR by using duplicate observations to retrieve primary observations (i.e., our queries are duplicate observations and our corpus consists of primary TR observations).

4.1.3 Bi-Encoder Training Details

The MS MARCO bi-encoder model is fine-tuned for four epochs on 45k randomly selected query, document pairs. The training is done through MultipleNegativeRanking loss function [*], where it is assumed that for all input pairs [(q_0. d_0), (q_1. d_1), ..., (q_i. d_i), ..., (q_N. d_N)], (q_i, d_i) represents a query and document which are similar (i.e., positive) and all other pairs (q_i, d_j) are dissimilar (i.e., negative). The learning rate used is 10^{-5}.

When fine-tuning the bi-encoder (TeleRoBERTa or MS MARCO) on the TR data, the model is trained for 10 epochs with a learning rate of 10^{-5}. MultipleNegativeRanking loss is used as well.

4.1.4 Cross-Encoder Training Details

The MS MARCO cross-encoder model is fine-tuned for four epochs with a learning rate of $2 * 10^{-5}$ on a subset of the MS MARCO dataset. We select 20k datapoints from MS MARCO and, for each query, we provide the model one

[*] https://www.sbert.net/docs/package_reference/losses.html

positive document example (i.e., the associated document) and three randomly selected negative documents. The model is trained through Binary Cross Entropy loss [*].

When fine-tuning the cross-encoder on the TR data, we again train for four epochs with a learning rate of $2 * 10^{-5}$. Again, we employ a 1:3 ratio for positive and negative samples.

4.1.5 EWC Hyperparameters

The standard MS w/ EWC model is fine-tuned with $\lambda = 10^6$ as we found the best and most stable results under this hyperparameter value.

The remainder of this chapter is organized as follows: First, we show the initial retrieval results attained by the different bi-encoder models. Second, we show the re-ranker results under evaluation scenarios 1 and 4 with all bi-encoder and cross-encoder combinations. Finally, we show the performance of models when applied on an out-of-domain TR dataset.

Fine-Tuning Strategy Acronyms

In the following section, we will sometimes refer to the different fine-tuning strategies with the following acronyms:

1. DA: Domain Adaptation
2. SL: Sequential Learning
3. MS: Our multi-stage fine-tuning approach
4. MS w/ EWC: Our multi-stage fine-tuning approach with EWC (lambda=10^6)

4.2 Bi-Encoder (Initial Retrieval) Results

4.2.1 Transferability of TeleRoBERTa to the TR domain

An important step is to identify if there is relevant telecommunications knowledge within the TeleRoBERTa model that will help us on the

[*] https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

downstream TR task. We run three experiments where we directly compare a TeleRoBERTa and RoBERTa model to identify if this is the case.

Experiment 1: Direct Application of Models

First, we fine-tune a TeleRoBERTa and RoBERTa model on the TR data directly. As shown in Table 4.2, we observe a near negligible difference between the two models when directly fine-tuned on the TR dataset. This provides an initial indication that the transferability of TeleRoBERTa to the TR dataset may be limited.

Table 4.2: Experiment 1: Does TeleRoBERTa transfer better to TR data than RoBERTa, when we directly fine-tune the model on the TR data/task?

Metrics	RoBERTa	TeleRoBERTa
Recall@1	10.28%	10.85%
Recall@3	20.77%	19.84%
Recall@5	26.25%	25.85%
Recall@10	35.32%	34.27%
Recall@15	32.18%	41.53%
Recall@20	26.41%	45.85%
MRR@5	16.04%	15.96%
MRR@15	17.78%	17.62%

Experiment 2: Zero-shot with MS MARCO

The previous experiment indicated limited difference between RoBERTa and TeleRoBERTa when fine-tuned on the TR data. However, this does not necessarily indicate that there is no initial greater transferability of the TeleRoBERTa model to the TR data compared to RoBERTa, as it is possible that the differences between the model performances are mitigated when fine-tuning on the TR data. To evaluate the initial transferability of the models, we set up a zero-shot experiment (i.e., direct application on the target domain/task with no explicit fine-tuning on it).

As RoBERTa and TeleRoBERTa are not fine-tuned to produce meaningful sentence/document embeddings, we initially fine-tune the models on MS MARCO data for 4 epochs. After which, we directly evaluate the model performance on the TR evaluation data, the results of which can be seen in Table 4.3.

Table 4.3: Experiment 2: Does TeleRoBERTa transfer better to TR data than RoBERTa, when we fine-tune the model on MS MARCO (i.e., fine-tune on the target task, but not domain)?

Metrics	RoBERTa	TeleRoBERTa
Recall@1	1.33%	2.42%
Recall@3	2.50%	4.40%
Recall@5	3.27%	5.32%
Recall@10	5.28%	7.30%
Recall@15	6.37%	9.15%
Recall@20	7.18%	10.16%
MRR@5	1.98%	3.48%
MRR@15	2.33%	3.87%

Despite the limited transferability outlined in the first experiment, we find that in a zero-shot context TeleRoBERTa significantly outperforms RoBERTa when applied to the TR data.

Experiment 3: Multi-stage fine-tuning

To further investigate model transferability, we fine-tune the models from Experiment 2 on our TR data. As can be observed in Table 4.4, we again see that the difference between the model performances is minimal. Notably, however, we do see a marginal improvement in both models in comparison to experiment 1, indicating that additional fine-tuning on MS MARCO may lead to stronger overall results.

Table 4.4: Experiment 3: Does TeleRoBERTa transfer better to TR data than RoBERTa, when we fine-tune the model on MS MARCO (i.e., fine-tune on the target task) and then fine-tune on TR data?

Metrics	RoBERTa	TeleRoBERTa
Recall@1	10.28%	11.45%
Recall@3	20.77%	21.05%
Recall@5	26.25%	26.09%
Recall@10	25.32%	34.76%
Recall@15	42.18%	41.57%
Recall@20	26.41%	46.01%
MRR@5	16.04%	16.73%
MRR@15	17.78%	18.40%

4.2.2 Comparison of Fine-Tuning Approaches

As outlined in the Methods section, we aim to investigate the impact of several fine-tuning strategies, notably domain adaptation, sequential learning, and our multi-stage fine-tuning strategy.

The performance of these different fine-tuning strategies is outlined in Table 4.5. We observe a non-negligible difference between sequential learning and the other strategies, with sequential learning performing the worst out of all strategies. However, there does not appear to be an immediate benefit to the multi-stage fine-tuning strategy over the simpler domain adaptation approach.

Table 4.5: Experiment 4: How do the different fine-tuning strategies perform on our TR data task?

Metrics	Domain Adaptation	Sequential Learning	Multi-Stage Fine-tuning
Recall@1	10.28%	10.85%	11.45%
Recall@3	20.77%	19.84%	21.05%
Recall@5	26.25%	25.85%	26.09%
Recall@10	35.32%	34.27%	34.76%
Recall@15	42.18%	41.53%	41.57%
Recall@20	46.41%	45.85%	46.01%
MRR@5	16.04%	15.96%	16.73%
MRR@15	17.78%	17.62%	18.40%

4.2.3 Impact of Catastrophic Forgetting Mitigation Strategies

A potential reason for why a multi-stage fine-tuning approach may not outperform the simpler domain adaptation strategy is due to catastrophic forgetting. i.e., As the TeleRoBERTa model fine-tunes on MS MARCO data, it begins to forget the telecommunications-specific knowledge that may have aided performance on the final task.

To mitigate this potential scenario, we adapt the multi-stage fine-tuning approach to use EWC when we fine-tune TeleRoBERTa on MS MARCO. We investigate EWC with 3 different hyperparameter setups, notably when λ is 10^4, 10^6, and 10^7. The results of which can be observed in Table 4.6.

Table 4.6: Experiment 5: Does EWC improve our final performance on the TR duplicate retrieval task? Note that the numbers next the EWC columns indicate the lambda hyperparameter value.

Metrics	No EWC	EWC (10^4)	EWC (10^6)	EWC (10^7)
Recall@1	11.45%	10.73%	11.01%	10.89%
Recall@3	21.05%	19.35%	20.28%	20.52%
Recall@5	26.09%	24.52%	25.12%	26.49%
Recall@10	34.76%	33.51%	35.73%	34.44%
Recall@15	41.57%	39.84%	40.97%	40.77%
Recall@20	46.01%	44.31%	45.24%	45.16%
MRR@5	16.73%	15.62%	16.10%	16.38%
MRR@15	18.40%	17.32%	17.93%	17.91%

4.2.4 Model Performance in Different Evaluation Scenarios

In Table 4.7, we evaluate the bi-encoders trained through the different fine-tuning processes on the four evaluation scenarios outlined in subsection 4.1.2. Note the significant increase in model performance when we use primary TR observations to retrieve the primary TR answer (Scenario 3) compared to when we use duplicate TR observations (Scenario 1). This indicates that there is shared information between the observation and answer section of a TR that may not be relevant to a true fault description, indicating that Scenario 2 and 3 may be biased and non-generalizable evaluation approaches. The most interested result can be seen in Scenario 4, however. In this case, the model is evaluated on its ability to retrieve a primary TR observation when given a duplicate TR observation as a query. Despite the model not having been trained on this task directly, we find a major performance increase when shifting to this form of data. These results indicate that there is far more useful information shared between a duplicate and primary TRs' observation sections than between their observation and answer sections.

4.2.5 Does EWC mitigate catastrophic forgetting?

Although the previous experiment investigated the final, downstream performance on the TR ranking task, it did not directly measure the impact of mitigating catastrophic forgetting. As outlined in the background of this thesis, there are several metrics that can be used to identify the severity of catastrophic forgetting in a model, e.g., BWT. These metrics generally rely on

Table 4.7: MRR@5 of each bi-encoder under the four evaluation scenarios. Note that DA refers to domain adapation, SL to sequential learning, MS to our multi-stage fine-tuning approach, and MS w/ EWC to our multi-stage fine-tuning approach with EWC using a lambda of 10^6.

Model	Scenario 1	Scenario 2	Scenario 3	Scenario 4
DA	16.04%	19.98%	27.89%	27.59%
SL	15.96%	19.91%	27.85%	26.64%
MS	16.73%	20.86%	29.14%	27.58%
MS w/ EWC	16.10%	20.86%	30.39%	26.99%

accuracy, which is not easily available for our dynamic MLM task, hence we instead investigate the change in loss when applying TeleRoBERTa on a subset of the data it was trained on through dynamic masked language modelling before and after fine-tuning on MS MARCO.

Table 4.8: Average loss per batch (over 871 batches) of a model on a subset of the original TeleRoBERTa training data. We show the loss before and after fine-tuning TeleRoBERTa on MS MARCO, with and without EWC.

	Before Fine-Tuning	No EWC	With EWC (10^6)
Average Loss per Batch	0.649	1.990	1.004

4.3 Re-Ranking Results

Similarly to the section 4.2, we fine-tune four cross-encoder models for each of the fine-tuning strategies, i.e., domain adaption (DA), sequential learning (SL), multi-stage fine-tuning (MS), and multi-stage fine-tuning with EWC (MS w/ EWC).

In Tables 4.9 and 4.10, we evaluate the combination of all bi and cross-encoder models on evaluation scenarios 1 and 4, respectively. In Table 4.9, we find that the sequential learning (SL) approach performs far better than domain adaptation and the multi-stage fine-tuning strategies, indicating that prior telecommunications knowledge aids the model performance. This hypothesis is further supported by observing that the cross-encoder fine-tuned with EWC (CE MS w/ EWC) attains similar performance to the sequential learning model.

In Table 4.10, we again observe that the sequential learning and MS w/ EWC cross-encoders attain the highest performance. However, in this case the

Table 4.9: Re-ranker results with MRR@5 using different bi-encoders and cross-encoders. Evaluation is done under Scenario 1 (i.e., queries consisting of duplicate TR observations and the corpus consisting of primary TR answers). Note that BE stands for Bi-Encoder and CE stands for Cross-Encoder. By Multi-Stage, we are referring to our multi-stage fine-tuning approach.

Models	CE DA	CE SL	CE MS	CE MS w/ EWC
BE DA	21.28%	22.98%	21.19%	22.78%
BE SL	21.60%	23.80%	21.54%	23.09%
BE MS	21.87%	23.65%	21.80%	23.72%
BE MS w/ EWC	21.49%	23.73%	21.77%	23.34%

MS w/ EWC cross-encoder outperforms the sequential learning cross encoder over all bi-encoder scenarios, indicating that the integration of MS MARCO into the fine-tuning process aided model performance. We also observe a more significant performance increase when using a multi-stage fine-tuning strategy over just domain adaptation.

Table 4.10: Re-ranker results with MRR@5 using different bi-encoders and cross-encoders. Evaluation is done under Scenario 4 (i.e., queries consisting of duplicate TR observations and the corpus consisting of primary TR observations). Note that BE stands for Bi-Encoder and CE stands for Cross-Encoder. By Multi-Stage, we are referring to our multi-stage fine-tuning approach.

Models	CE DA	CE SL	CE MS	CE MS w/ EWC
BE DA	35.67%	38.81%	37.99%	39.48%
BE SL	35.43%	38.22%	37.47%	38.62%
BE MS	36.06%	38.98%	38.06%	39.21%
BE MS w/ EWC	36.04%	39.31%	38.50%	39.45%

4.4 Generalizability to other datasets

Thus far, we have evaluated our model on a single dataset in which the trouble reports (train, validation, and test) were sourced from a single domain/group of trouble reports. In this subsection, we aim to identify the performance drop, if any, when evaluating the pretrained models on an out-of-domain TR dataset, notably where we are focusing on TRs created by a different set of operators. We evaluate over 3.4K TRs, of which 1.2K are primary TRs and 2.2K are duplicate TRs.

In Tables 4.11 and 4.12, the full re-ranking results are shown for evaluation scenarios 1 and 4, respectively. Although we do observe a decrease in performance compared to the results outlined in section 4.3, this drop is within acceptable bounds. Especially for scenario 4, we only observe a 3.5% drop in MRR@5 performance. Interestingly, in this scenario we observe that the multi-stage re-ranking process (i.e., using both a bi-encoder and cross-encoder fine-tuned in our multi-stage fine-tuning process) achieves equivalent and even slightly better performance than the sequential learning and MS w/ EWC re-ranking approaches under evaluation scenario 4, despite a larger gap having been observed in Section 4.3.

Table 4.11: Results on the out-of-domain TR dataset with Scenario 1 evaluation (i.e., queries consisting of duplicate TR observations and the corpus consisting of primary TR answers). Note that the models used are bi-encoder/cross-encoder pairs (e.g., DA uses both the domain adaptation bi-encoder and cross-encoder)

Metrics	DA	SL	MS	MS w/ EWC
Recall@1	10.96%	13.63%	12.53%	13.50%
Recall@3	18.70%	21.83%	22.02%	22.43%
Recall@5	23.26%	25.98%	26.02%	26.44%
Recall@10	30.17%	32.70%	31.37%	33.86%
Recall@15	35.15%	36.71%	36.11%	37.36%
Recall@20	37.54%	39.29%	40.40%	40.17%
MRR@5	15.40%	18.16%	17.64%	18.31%
MRR@15	16.69%	19.37%	18.73%	19.59%

4.5 Discussion

At the start of this thesis, we outlined three key research questions and goals. In this section, we provide a discussion on the insight the results give to our these questions and goals and provide any additional thoughts/insight gained from the analyses.

4.5.1 Comparison of Transfer Learning Strategies

Somewhat surprisingly, we found virtually no major difference between the performance of the bi-encoder models. Even though EWC clearly did mitigate

Table 4.12: Results on the out-of-domain TR dataset with Scenario 4 evaluation (i.e., queries consisting of duplicate TR observations and the corpus consisting of primary TR answers). Note that the models used are bi-encoder/cross-encoder pairs (e.g., DA uses both the domain adaptation bi-encoder and cross-encoder)

Metrics	DA	SL	MS	MS w/ EWC
Recall@1	23.35%	28.42%	28.01%	28.14%
Recall@3	37.95%	42.38%	42.84%	42.47%
Recall@5	43.85%	47.86%	48.83%	48.96%
Recall@10	50.90%	53.11%	54.31%	54.12%
Recall@15	54.72%	56.20%	57.99%	57.35%
Recall@20	56.89%	58.27%	60.29%	59.19%
MRR@5	31.18%	35.83%	36.00%	35.95%
MRR@15	32.45%	36.81%	37.03%	36.92%

catastrophic forgetting, as can be observed in Table 4.8, different values of EWC did not have a major impact on the results. These findings are in stark contrast to the full re-ranking performance, where we found a more substantial difference between the models. Notably, the SL and MS w/ EWC cross-encoders consistently outperformed the DA and MS cross-encoders, both when evaluating under scenario 1 and scenario 4 as can be seen in Tables 4.9 and 4.10, respectively. This indicates that integrating telecommunications-specific data in the fine-tuning process did provide a substantial benefit in overall model performance. The results in scenario 4, in Table 4.10, show that the MS w/ EWC cross encoder marginally attained the strongest results. Although the margins are too small to conclude that the multi-stage fine-tuning strategy improved over a sequential learning fine-tuning strategy, it does provide a positive outlook over the possibilities of continuous learning in NLP fine-tuning strategies.

4.5.2 Multi-Stage Fine-Tuning with EWC

As demonstrated in Table 4.8, we find that EWC does mitigate catastrophic forgetting substantially. We also observe a benefit in the re-ranking results when we use EWC in the multi-stage fine-tuning approach. The results, however, did not indicate a significant improvement over other, simpler transfer learning strategies. Hence further investigation will need to take place to identify if there are benefits to a multi-stage fine-tuning strategy.

4.5.3 Generalizability to Other Domains

Although we do see a clear drop in performance when evaluating the re-ranking framework on an out-of-domain dataset, as can be seen when comparing the MRR@5 values when comparing Tables 4.11 and 4.12 to Tables 4.9 and 4.10, we still observe strong model performance overall. All models other than the DA bi/cross-encoder pair attained near equal performance on the new dataset, with the MS models attaining similar results to SL and MS w/ EWC.

Chapter 5

Conclusions and Future work

5.1 Conclusions

The trouble reporting process at Ericsson is used to identify, report, analyze, and eventually resolve software and hardware faults. Due to the scale of the organization, however, we often find duplicate trouble reports that, if not identified, represent a significant amount of unnecessary additional effort to resolve. In this thesis, we investigate and evaluated how four fine-tuning strategies impacted the performance of RoBERTa-based models for retrieving duplicate trouble reports.

We find that integrating existing telecommunications knowledge through the form of a pretrained telecommunications-specific language model into our fine-tuning strategies allows us to outperform a domain adaptation fine-tuning strategy. In addition to this, we find that EWC is an effective strategy for mitigating catastrophic forgetting and attaining strong downstream performance on the duplicate TR retrieval task.

Finally, we find that the generalizability of models is strong enough to perform reasonably effectively on out-of-domain TR data, indicating that the approaches outlined in this thesis may be eligible in a real-world deployment.

5.2 Limitations

There are several notable limitations to this thesis, most notably in regards to the generalizability of the insights in this study to domains other than TRs and telecommunications. Some additional limitations we identified are as follows:

- We only applied one main catastrophic forgetting mitigation strategy in

EWC. Hence, we cannot say that all catastrophic forgetting mitigation strategies will lead to similar results we observed. It is possible that with different mitigation strategies we would see higher or lower performance for our multi-stage fine-tuning models.

- Due to computational limitations in the amount of models that could be trained, we cannot run a statistical analysis to identify if models consistently outperform each other in the way that we would expect them to. For the most part, each model were trained to completion only one or two times which may lead to some bias in the final results.
- The evaluation dataset size was somewhat limited. Evaluating on 10k+ duplicate TRs may significantly change how the results look.
- As our multi-stage fine-tuning strategy is a novel approach to fine-tuning, our work may have implications for general transfer learning research in large language models. However, due to the limitations in the domain of our data (TR/telecommunications data) we cannot make more general conclusions.
- Although it is clear that the TeleRoBERTa model had some domain overlap with our TR data, it did not provide a major benefit in many cases. Evaluating the same fine-tuning schema with a model that has near perfect domain overlap with the final downstream task may be interesting to investigate.

5.3 Future work

Due to the scale of this thesis, there are many future directions to explore. What may be most interesting is to observe this form of fine-tuning in domains other than telecommunications to see if similar insights hold. Thus far, transfer learning in NLP lacks clear standards for which strategy to employ in which scenario, especially in more complex tasks such as duplicate TR retrieval. Additional experimental research on fine-tuning strategies over a large range of domains and tasks could begin to lay the foundation for these sorts of standards.

More specific to the content of this thesis, there is much to explore in regards to the hyperparameter tuning and learning rate scheduling for mitigating catastrophic forgetting and attaining stronger overall modelling results [43, 42].

In addition to this, one of the core elements of EWC lies in the Fisher Information Matrix, where parameter-wise importance is stored. It would be very interesting to see how computing the fisher information matrix on different subsets of data may impact the parameter-wise importance estimates.

Finally, multi-task learning as a transfer learning strategy was, in large part, ignored in this thesis. It may, however, be a very strong approach to building robust models that perform well on multiple tasks simultaneously. Hence, it would be worthwhile to evaluate multi-task learning as another fine-tuning strategy.

References

[1] N. Marzo i Grimalt, "Natural language processing model for log analysis to retrieve solutions for troubleshooting processes," 2021. [Pages i, iii, ix, 1, 2, 3, and 4.]

[2] H. Holm, "Bidirectional encoder representations from transformers (bert) for question answering in the telecom domain.: Adapting a bert-like language model to the telecom domain using the electra pre-training approach," 2021. [Pages i, iii, 3, and 38.]

[3] L. Jonsson, *Machine Learning-Based Bug Handling in Large-Scale Software Development*. Linköping University Electronic Press, 2018, vol. 1936. [Page 1.]

[4] N. M. I. Grimalt, S. Shalmashi, F. Yaghoubi, L. Jonsson, and A. H. Payberah, "Berticsson: A recommender system for troubleshooting," 2022. [Pages 1, 23, 36, 41, and 44.]

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018. [Pages 2, 15, and 24.]

[6] S. Robertson and H. Zaragoza, *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009. [Pages 2 and 18.]

[7] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of learning and motivation*. Elsevier, 1989, vol. 24, pp. 109–165. [Pages 5 and 28.]

[8] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999. [Page 5.]

[9] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," *arXiv preprint arXiv:1312.6211*, 2013. [Page 5.]

[10] S. Ruder, "Neural transfer learning for natural language processing," Ph.D. dissertation, NUI Galway, 2019. [Pages xi, 5, 25, 27, 28, and 67.]

[11] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017. [Pages 6, 7, 28, 29, 30, and 67.]

[12] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *arXiv preprint arXiv:1906.02243*, 2019. [Page 7.]

[13] F. M. Zanzotto, "Human-in-the-loop artificial intelligence," *Journal of Artificial Intelligence Research*, vol. 64, pp. 243–252, 2019. [Pages 7 and 8.]

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. [Pages 10, 11, and 15.]

[15] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014. [Page 10.]

[16] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014. [Page 11.]

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. [Page 13.]

[18] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016. [Page 16.]

[19] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, “Albert: A lite bert for self-supervised learning of language representations,” *arXiv preprint arXiv:1909.11942*, 2019. [Page 17.]

[20] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, “Roberta: A robustly optimized bert pretraining approach,” *arXiv preprint arXiv:1907.11692*, 2019. [Pages 17 and 38.]

[21] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, “Electra: Pre-training text encoders as discriminators rather than generators,” *arXiv preprint arXiv:2003.10555*, 2020. [Page 17.]

[22] C. Sammut and G. I. Webb, Eds., *TF–IDF*. Boston, MA: Springer US, 2010, pp. 986–987. ISBN 978-0-387-30164-8. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_832 [Page 18.]

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013. [Page 18.]

[24] J. Lin, R. Nogueira, and A. Yates, “Pretrained transformers for text ranking: Bert and beyond,” *Synthesis Lectures on Human Language Technologies*, vol. 14, no. 4, pp. 1–325, 2021. [Pages 18 and 19.]

[25] J. Guo, Y. Fan, L. Pang, L. Yang, Q. Ai, H. Zamani, C. Wu, W. B. Croft, and X. Cheng, “A deep look into neural ranking models for information retrieval,” *Information Processing & Management*, vol. 57, no. 6, p. 102067, 2020. [Page 18.]

[26] M. Trabelsi, Z. Chen, B. D. Davison, and J. Heflin, “Neural ranking models for document retrieval,” *Information Retrieval Journal*, vol. 24, no. 6, pp. 400–444, 2021. [Page 19.]

[27] J. Mueller and A. Thyagarajan, “Siamese recurrent architectures for learning sentence similarity,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016. [Page 19.]

[28] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. Ward, “Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 4, pp. 694–707, 2016. [Page 19.]

[29] L. Pang, Y. Lan, J. Guo, J. Xu, J. Xu, and X. Cheng, “Deeprank: A new deep architecture for relevance ranking in information retrieval,” in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 257–266. [Page 19.]

[30] Z. Dai, C. Xiong, J. Callan, and Z. Liu, “Convolutional neural networks for soft-matching n-grams in ad-hoc search,” in *Proceedings of the eleventh ACM international conference on web search and data mining*, 2018, pp. 126–134. [Page 19.]

[31] N. Reimers and I. Gurevych, “Sentence-bert: Sentence embeddings using siamese bert-networks,” *arXiv preprint arXiv:1908.10084*, 2019. [Pages 21 and 22.]

[32] D. Chicco, “Siamese neural networks: An overview,” *Artificial Neural Networks*, pp. 73–94, 2021. [Page 22.]

[33] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, “Dense passage retrieval for open-domain question answering,” *arXiv preprint arXiv:2004.04906*, 2020. [Page 22.]

[34] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. Bennett, J. Ahmed, and A. Overwijk, “Approximate nearest neighbor negative contrastive learning for dense text retrieval,” *arXiv preprint arXiv:2007.00808*, 2020. [Page 22.]

[35] K. Lee, M.-W. Chang, and K. Toutanova, “Latent retrieval for weakly supervised open domain question answering,” *arXiv preprint arXiv:1906.00300*, 2019. [Page 22.]

[36] R. Nogueira, W. Yang, K. Cho, and J. Lin, “Multi-stage document ranking with bert,” *arXiv preprint arXiv:1910.14424*, 2019. [Pages 22 and 23.]

[37] O. Khattab and M. Zaharia, “Colbert: Efficient and effective passage search via contextualized late interaction over bert,” in *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, 2020, pp. 39–48. [Page 22.]

[38] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009. [Pages xi, 24, and 25.]

[39] S. Thrun, "Lifelong learning algorithms," in *Learning to learn*. Springer, 1998, pp. 181–209. [Page 26.]

[40] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith, "Don't stop pretraining: adapt language models to domains and tasks," *arXiv preprint arXiv:2004.10964*, 2020. [Page 26.]

[41] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi, and N. Smith, "Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping," *arXiv preprint arXiv:2002.06305*, 2020. [Page 26.]

[42] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" in *China national conference on Chinese computational linguistics*. Springer, 2019, pp. 194–206. [Pages 26, 27, 28, 32, 56, and 67.]

[43] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018. [Pages 26, 28, 32, and 56.]

[44] A. Rogers, O. Kovaleva, and A. Rumshisky, "A primer in bertology: What we know about how bert works," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 842–866, 2020. [Page 26.]

[45] I. Tenney, D. Das, and E. Pavlick, "Bert rediscovers the classical nlp pipeline," *arXiv preprint arXiv:1905.05950*, 2019. [Page 26.]

[46] Y. Lin, Y. C. Tan, and R. Frank, "Open sesame: getting inside bert's linguistic knowledge," *arXiv preprint arXiv:1906.01698*, 2019. [Page 26.]

[47] O. Kovaleva, A. Romanov, A. Rogers, and A. Rumshisky, "Revealing the dark secrets of bert," *arXiv preprint arXiv:1908.08593*, 2019. [Page 26.]

[48] N. Durrani, H. Sajjad, and F. Dalvi, "How transfer learning impacts linguistic knowledge in deep nlp models?" *arXiv preprint arXiv:2105.15179*, 2021. [Page 26.]

[49] A. Merchant, E. Rahimtoroghi, E. Pavlick, and I. Tenney, "What happens to bert embeddings during fine-tuning?" *arXiv preprint arXiv:2004.14448*, 2020. [Page 26.]

[50] M. Mosbach, A. Khokhlova, M. A. Hedderich, and D. Klakow, “On the interplay between fine-tuning and sentence-level probing for linguistic knowledge in pre-trained transformers,” *arXiv preprint arXiv:2010.02616*, 2020. [Page 26.]

[51] J. Wallat, J. Singh, and A. Anand, “Bertnesia: Investigating the capture and forgetting of knowledge in bert,” *arXiv preprint arXiv:2106.02902*, 2021. [Page 26.]

[52] J. Lovón-Melgarejo, L. Soulier, K. Pinel-Sauvagnat, and L. Tamine, “Studying catastrophic forgetting in neural ranking models,” *arXiv preprint arXiv:2101.06984*, 2021. [Pages 27 and 31.]

[53] G. M. Van de Ven and A. S. Tolias, “Three scenarios for continual learning,” *arXiv preprint arXiv:1904.07734*, 2019. [Page 27.]

[54] R. Ratcliff, “Connectionist models of recognition memory: constraints imposed by learning and forgetting functions.” *Psychological review*, vol. 97, no. 2, p. 285, 1990. [Page 28.]

[55] D. Lopez-Paz and M. Ranzato, “Gradient episodic memory for continual learning,” *Advances in neural information processing systems*, vol. 30, 2017. [Pages 28 and 31.]

[56] N. Díaz-Rodríguez, V. Lomonaco, D. Filliat, and D. Maltoni, “Don’t forget, there is more than forgetting: new metrics for continual learning,” *arXiv preprint arXiv:1810.13166*, 2018. [Page 29.]

[57] F. Zenke, B. Poole, and S. Ganguli, “Continual learning through synaptic intelligence,” in *International Conference on Machine Learning*. PMLR, 2017, pp. 3987–3995. [Pages 29, 31, and 67.]

[58] R. Pascanu and Y. Bengio, “Revisiting natural gradient for deep networks,” *arXiv preprint arXiv:1301.3584*, 2013. [Pages 30 and 39.]

[59] Y. Xu, X. Zhong, A. J. J. Yepes, and J. H. Lau, “Forget me not: Reducing catastrophic forgetting for domain adaptation in reading comprehension,” in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8. [Page 31.]

[60] J. Thorne and A. Vlachos, “Elastic weight consolidation for better bias inoculation,” *arXiv preprint arXiv:2004.14366*, 2020. [Page 31.]

[61] P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, A. McNamara, B. Mitra, T. Nguyen *et al.*, “Ms marco: A human generated machine reading comprehension dataset,” *arXiv preprint arXiv:1611.09268*, 2016. [Page 38.]

[62] G. E. Hinton and S. Roweis, “Stochastic neighbor embedding,” *Advances in neural information processing systems*, vol. 15, 2002. [Page 68.]

Appendix A

Additional Results

A.1 Simultaneous Learning

In this thesis we have, for the most part, focused on sequential learning of tasks and domains. However, as outlined by [10], learning tasks simultaneously is also a viable option for transfer learning in NLP. By learning tasks simultaneously, catastrophic forgetting of tasks does not occur to a significant extent, as the model is frequently provided data associated to a previous task [11, 57].

To investigate the eligibility of this approach, we constructed the following experiment: Fine-tune a model on 10k datapoints of MS MARCO and the TR dataset simultaneously (all data shuffled) and evaluate how this impacts the performance on duplicate TR retrieval.

This experiment was only performed with the bi-encoder. We evaluated this by training for 10 epochs and using three different learning rates: 10^{-4}, 10^{-5}, and 10^{-6}.

The results can be found in tables A.1 and A.2, which outline the results for Scenario 1 and Scenario 4 evaluation. We find that a learning rate of 10^{-5} performs the best. The results do not outperform the models outlined in section 4.2, however. There are several reasons for why this might be the case: 1) Only 10k datapoints from MS MARCO are used, compared to 45k in section 4.2, 2) we are only investigating initial retrieval, whereas we may see more interesting results when investigating cross encoders, and 3) we do not perform an additional fine-tuning step on only TR data to further optimize model performance, which may be beneficial [42].

Table A.1: Simultaneous Learning Results with a Bi-Encoder, Scenario 1 Evaluation

Metrics	lr=10^{-4}	lr=10^{-5}	lr=10^{-6}
Recall@1	0.00%	9.15%	8.39%
Recall@3	0.24%	16.94%	14.52%
Recall@5	0.36%	23.10%	17.94%
Recall@10	0.85%	32.58%	24.80%
Recall@15	1.05%	39.23%	29.60%
Recall@20	1.49%	44.35%	33.27%
MRR@5	0.12%	13.94%	11.83%
MRR@15	0.20%	15.71%	13.12%

Table A.2: Simultaneous Learning Results with a Bi-Encoder, Scenario 4 Evaluation

Metrics	lr=10^{-4}	lr=10^{-5}	lr=10^{-6}
Recall@1	0.05%	17.66%	17.18%
Recall@3	0.36%	28.02%	25.56%
Recall@5	0.44%	34.19%	30.40%
Recall@10	0.77%	43.23%	39.07%
Recall@15	1.09%	49.35%	44.80%
Recall@20	1.65%	54.56%	48.39%
MRR@5	0.19%	23.63%	21.96%
MRR@15	0.26%	25.31%	23.56%

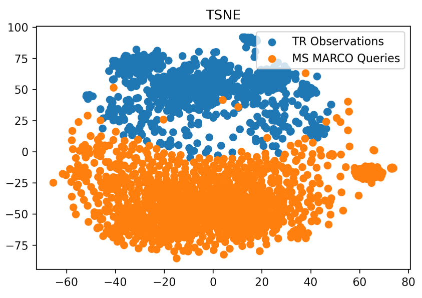
A.2 Visualizing Model Embeddings

As the output of a bi-encoder model is an embedding (i.e., fixed length vector output from the BERT model) of an input document, it may be interesting to visualize how these embeddings look for different datasets and at different stages in the fine-tuning process. To perform this visualization effectively, we reduce the dimensionality of the embeddings to 2 dimensions using t-SNE [62] [*].

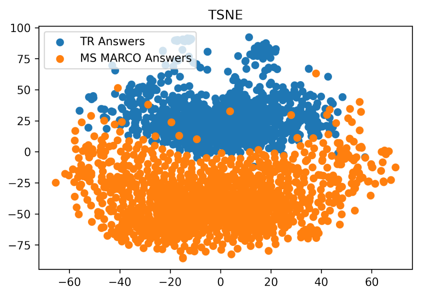
In Figure A.1, we show how the 2D embeddings of a RoBERTa bi-encoder fine-tuned on MS MARCO change when we fine-tune it on the TR data. Figures A.2 demonstrates the same analysis, but for TeleRoBERTa fine-tuned on MS MARCO and TR data. Finally, in Figure A.3 we visualize the same analysis, but with EWC loss leveraged in the fine-tuning process.

[*] PCA was also used, but we did not find a substantial difference in results hence we only report t-SNE

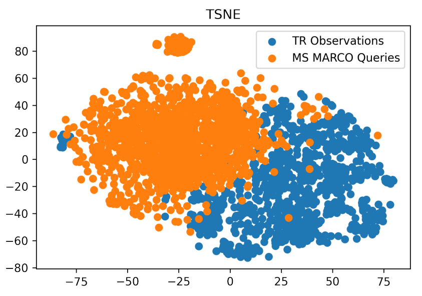
Naturally, the 2 dimensional representation is unlikely to show all the interesting elements of the real high dimensional embeddings, but we do see that by fine-tuning on TR data we see greater similarity on the embeddings of MS MARCO and TR data together.



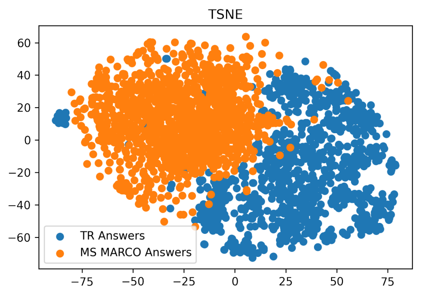
(a) TR Observations and MS MARCO Queries



(b) TR Answers and MS MARCO Answers/Documents

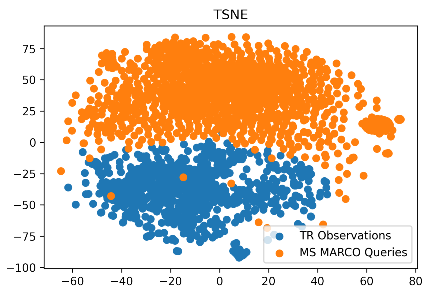


(c) TR Observations and MS MARCO Queries

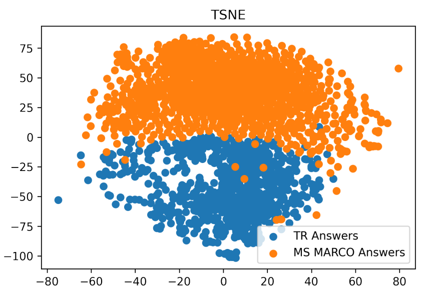


(d) TR Answers and MS MARCO Answers/Documents

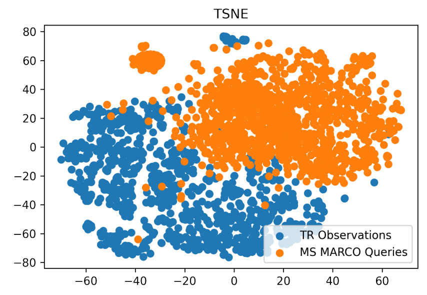
Figure A.1: RoBERTa embeddings on MS MARCO and TR data. The top row represents RoBERTa fine-tuned on 45k datapoints from MS MARCO. The bottom row is when that model has been fine-tuned on TR data. Note that we begin to see more overlap in the bottom row



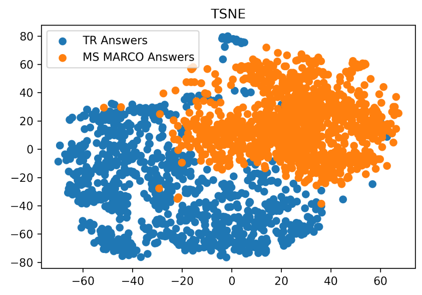
(a) TR Observations and MS MARCO Queries



(b) TR Answers and MS MARCO Answers/Documents

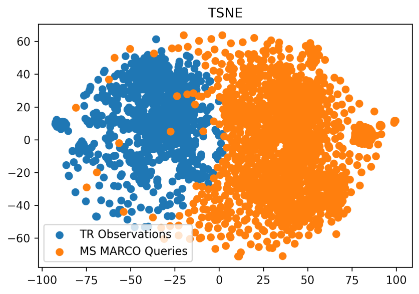


(c) TR Observations and MS MARCO Queries

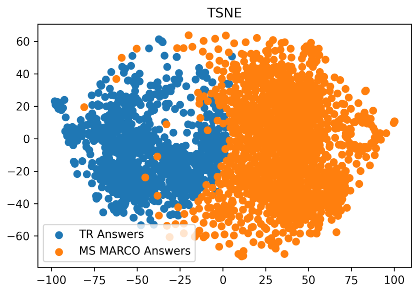


(d) TR Answers and MS MARCO Answers/Documents

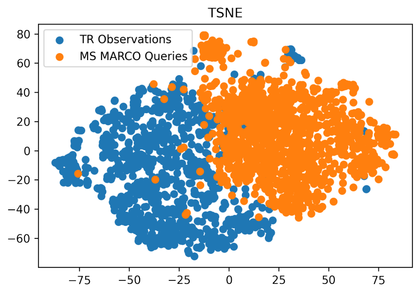
Figure A.2: TeleRoBERTa embeddings on MS MARCO and TR data. The top row represents TeleRoBERTa fine-tuned on 45k datapoints from MS MARCO. The bottom row is when that model has been fine-tuned on TR data. Note that we begin to see more overlap in the bottom row



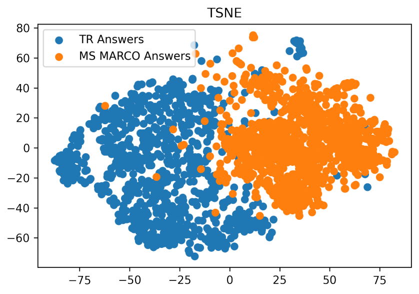
(a) TR Observations and MS MARCO Queries



(b) TR Answers and MS MARCO Answers/Documents



(c) TR Observations and MS MARCO Queries



(d) TR Answers and MS MARCO Answers/Documents

Figure A.3: TeleRoBERTa embeddings on MS MARCO and TR data when fine-tuning with EWC (lambda = 10^6). The top row represents TeleRoBERTa fine-tuned on 45k datapoints from MS MARCO. The bottom row is when that model has been fine-tuned on TR data. Note that we begin to see more overlap in the bottom row

TRITA-EECS-EX- 2022:00

www.kth.se

€€€€ For DIVA €€€€

{
”Author1”: { ”Last name”: ”Bosch”,
”First name”: ”Nathan G.”,
”E-mail”: ”nbosch@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
}
},
”Cycle”: ”2”,
”Course code”: ”DA233X”,
”Credits”: ”30.0”,
”Degree1”: {”Educational program”: ”Master’s Programme, Machine Learning, 120 credits”
,”programcode”: ”TMAIM”
,”Degree”: ”Master’s degree”
,”subjectArea”: ”Computer Science and Engineering, specializing in Machine Learning”
},
”Title”: {
”Main title”: ”Integrating Telecommunications-Specific Language Models into a Trouble Report Retrieval Approach”,
”Language”: ”eng” },
”Alternative title”: {
”Main title”: ”Integrering av telekommunikationsspecifika språkmodeller i en metod för hämtning av problemrapporter”,
”Language”: ”swe”
},
”Supervisor1”: { ”Last name”: ”Yaghoubi”,
”First name”: ”Forough”,
”E-mail”: ”forough.yaghoubi@ericsson.com”,
”Other organisation”: ”Ericsson AB, GFTL GAIA”
},
”Supervisor2”: { ”Last name”: ”Shalmashi”,
”First name”: ”Serveh”,
”E-mail”: ”serveh.shalmashi@ericsson.com”,
”Other organisation”: ”Ericsson AB, GFTL GAIA”
},
”Supervisor3”: { ”Last name”: ”Hakimzadeh”,
”First name”: ”Kamal”,
”Local User Id”: ”u1g0naq4”,
”E-mail”: ”kamal2@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
”L2”: ”Computer Science” }
},
”Examiner1”: { ”Last name”: ”Payberah”,
”First name”: ”Amir H.”,
”Local User Id”: ”u1a73o9d”,
”E-mail”: ”payberah@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
”L2”: ”Computer Science” }
},
”Cooperation”: { ”Partner_name”: ”Ericsson AB”},
"National Subject Categories": "10201, 10208, 20204",
”Other information”: {”Year”: ”2022”, ”Number of pages”: ”1,69”},
”Series”: { ”Title of series”: ”TRITA-EECS-EX” , ”No. in series”: ”2022:00” },
”Opponents”: { ”Name”: ”Fredrik Diffner”},
”Presentation”: { ”Date”: ”2022-06-16 14:00”
,”Language”:”eng”
,”Room”: ”via Zoom https://kth-se.zoom.us/j/2884945301”
,”Address”: ”Ada, Isafjordsgatan 22 (Kistagången 16)”
,”City”: ”Stockholm” },
”Number of lang instances”: ”2”,
”Abstract[eng]”: €€€€

In the development of large telecommunications systems, it is imperative to identify, report, analyze and, thereafter, resolve both software and hardware faults. This resolution process often relies on written trouble reports (TRs), that contain information about the observed fault and, after analysis, information about why the fault occurred and the decision to resolve the fault. Due to the scale and number of TRs, it is possible that a newly written fault is very similar to previously written faults, e.g., a duplicate fault. In this scenario, it can be beneficial to retrieve similar TRs that have been previously created to aid the resolution process.

Previous work at Ericsson \cite{marzo2021natural}, introduced a multi-stage BERT-based approach to retrieve similar TRs given a newly written fault observation. This approach significantly outperformed simpler models like BM25, but suffered from two major challenges: 1) it did not leverage the vast non-task-specific telecommunications data at Ericsson, something that had seen success in other work \cite{holm2021bidirectional}, and 2) the model did not generalize effectively to TRs outside of the telecommunications domain it was trained on.

In this thesis, we 1) investigate three different transfer learning strategies to attain stronger performance on a downstream TR duplicate retrieval task, notably focusing on effectively integrating existing telecommunications-specific language data into the model fine-tuning process, 2) investigate the efficacy of catastrophic forgetting mitigation strategies when fine-tuning the BERT models, and 3) identify how well the models perform on out-of-domain TR data.

We find that integrating existing telecommunications knowledge through the form of a pretrained telecommunications-specific language model into our fine-tuning strategies allows us to outperform a domain adaptation fine-tuning strategy. In addition to this, we find that Elastic Weight Consolidation (EWC) is an effective strategy for mitigating catastrophic forgetting and attaining strong downstream performance on the duplicate TR retrieval task. Finally, we find that the generalizability of models is strong enough to perform reasonably effectively on out-of-domain TR data, indicating that the approaches may be eligible in a real-world deployment.

€€€€,
”Keywords[eng]”: €€€€
information retrieval, neural ranking, trouble reports, log analysis, natural language processing €€€€,
”Abstract[swe]”: €€€€

Vid utvecklingen av stora telekommunikationssystem är det absolut nödvändigt att identifiera, rapportera, analysera och därefter lösa både mjukvaru och hårdvarufel. Denna lösningsprocess bygger ofta på noggrant skrivna felrapporter (TRs), som innehåller information om det observerade felet och, efter analys, information om varför felet uppstod och beslutet att åtgärda felet. På grund av skalan och antalet TR:er är det möjligt att ett nyskrivet fel är mycket likt tidigare skrivna fel, t.ex. ett duplikatfel. I det här scenariot kan det vara mycket fördelaktigt att hämta tidigare skapade, liknande TR:er för att underlätta upplösningsprocessen.

Tidigare arbete på Ericsson \cite{marzo2021natural}, introducerade en flerstegs BERT-baserad metod för att hämta liknande TRs givet en nyskriven felobservation. Detta tillvägagångssätt överträffade betydligt enklare modeller som BM-25, men led av två stora utmaningar: 1) det utnyttjade inte den stora icke-uppgiftsspecifika telekommunikationsdatan hos Ericsson, något som hade sett framgång i annat arbete \cite{holm2021bidirectional}, och 2) modellen generaliserades inte effektivt till TR:er utanför den telekommunikationsdomän som den bildades på.

I den här masteruppsatsen undersöker vi 1) tre olika strategier för överföringsinlärning för att uppnå starkare prestanda på en nedströms TR dubbletthämtningsuppgift, varav några fokuserar på att effektivt integrera fintliga telekommunikationsspecifika språkdata i modellfinjusteringsprocessen, 2) undersöker effektiviteten av katastrofala missglömningsreducerande strategier vid finjustering av BERT-modellerna, och 3) identifiera hur väl modellerna presterar på TR-data utanför domänen.

Resultatet är genom att integrera befintlig telekommunikationskunskap i form av en förtränad telekommunikationsspecifik språkmodell i våra finjusteringsstrategier kan vi överträffa en finjusteringsstrategi för domänanpassning. Utöver detta har vi fåt fram att EWC är en effektiv strategi för att mildra katastrofal glömska och uppnå stark nedströmsprestanda på dubbla TR hämtningsuppgiften. Slutligen finner vi att generaliserbarheten av modeller är tillräckligt stark för att prestera någorlunda effektivt på TR-data utanför domänen, vilket indikerar att tillvägagångssätten som beskrivs i denna avhandling kan vara kvalificerade i en verklig implementering.

€€€€,
”Keywords[swe]”: €€€€
informationssökning, neural rangordning, felrapporter, logganalys, naturlig språkbehandling €€€€,
}