TU WIEN Informatics

Recommending Reviewers for Theses using Artificial Intelligence

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

David Penz, BA
Matrikelnummer 11703497

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger

Wien, 25. Jänner 2021

David Penz	Horst Eidenberger

TU WIEN Informatics

Recommending Reviewers for Theses using Artificial Intelligence

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

David Penz, BA
Registration Number 11703497

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Mag. Dr. Horst Eidenberger

Vienna, 25th January, 2021

David Penz | Horst Eidenberger

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

David Penz, BA

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. Jänner 2021

David Penz

Acknowledgements

Throughout the writing of this thesis, I have received valuable support from many different people.

First, I want to thank my supervisor Horst Eidenberger for the possibility to work on one of his ideas as topic for my master thesis. Your insights and continued feedback over the complete duration of this process helped me shaping the content of this document.

Furthermore, I want to give my special thanks to Tamara Drucks. Thank you for all your emotional support, the many discussions about relevant topics and research, brainstorming sessions about potential ideas of mine, and your time for proofreading this thesis.

Additionally, I want to acknowledge all of my friends of the *BS Temple*, who supported me throughout various courses and projects, and pushed me with insightful discussions about all kinds of different topics. In particular, I want to thank Anna-Maria Nau, Giulio Pace, Matthias König, and Michael Bernreiter. I highly appreciate all your emotional support and guidance you have provided to me throughout this process.

Last but not least, I want to thank my parents and my sister. You have been supporting me for all of my life and continue to do so, which is invaluable to me.

Kurzfassung

Die Evaluierung wissenschaftlicher Arbeiten, wie zum Beispiel bei der Einreichung für eine Konferenz, spielt eine wichtige Rolle in der Wissenschaft, um deren Integrität und Korrektheit zu gewährleisten. Dies gilt auch für die Erlangung eines Masterabschlusses an der TU Wien, bei dem die Studierenden ihre Abschlussarbeit vor einer Prüfungskommission verteidigen müssen. Die Prüfungskommission besteht aus der Betreuerin oder dem Betreuer der Arbeit und zwei weiteren Prüferinnen oder Prüfern, die im Rahmen des Prozesses manuell ausgewählt werden müssen. Die manuelle Auswahl dieser Mitglieder ist anfällig für Fehleinschätzungen, was zu einer fehlerhaften Beurteilung während der Prüfung führen kann.

Ziel dieser Arbeit ist es, ein Programm für die Empfehlung der Mitglieder zu entwickeln, basierend auf aktuellen Methoden im Bereich der künstlichen Intelligenz (KI). Als Basis hierfür extrahieren wir notwendige Daten aus den internen Datenbanken der TU Wien. In weiterer Folge definieren wir verschiedene Architekturen für Neuronale Netze, welche anschließend mit Hilfe des Datensatzes trainiert werden. Die einzelnen Modelle basieren auf den Architekturen von *LSTMs*, *Autoencoder* und *Siamesischen Neuronalen Netzen*. Jedes Modell wird jeweils mit Hilfe von *BERT*, *GPT2* und *XLNet* als Basis trainiert. Anschließend evaluieren wir die Netze anhand der Aufgabe, potenzielle Forschungsprofile auf Basis einer Abschlussarbeit zu bewerten, und vergleichen die Ergebnisse mit BM25. Zusätzlich hierzu werden die Modelle anhand ausgewählter Abschlussarbeiten manuell validiert, was zu einer besseren Aussagekraft der Ergebnisse führt. Wir kommen zu der Schlussfolgerung, dass Architekturen basierend auf *Siamesischen Neuronalen Netzen* vielversprechende Ergebnisse erzielen und angewandt für die Re-Evaluierung von Profilen sogar BM25 übertreffen. Des Weiteren zeigen die Experimente, dass *BERT* als Basis die besten Resultate für alle Architekturen liefert. Abschließend schlussfolgern wir, dass der Auswahlprozess für die weiteren Prüferinnen und Prüfer optimiert werden kann, indem die von uns beschriebenen Methoden im Bereich der KI verwendet werden.

Abstract

Peer review in the area of scientific contributions, such as publishing papers to conferences, plays a crucial role to evaluate the integrity and correctness of the respective work. This is also the case for the process of obtaining a Master's degree at TU Wien, where students must defend their thesis in front of an examination board. The examination board consists of the supervisor and two additional reviewers, which must be selected manually as part of the process. The manual selection of those reviewers can be prone to human misjudgment, causing a faulty evaluation during the examination.

In this thesis, we aim to develop a recommendation engine driven by state-of-the-art methodology in the area of artificial intelligence. This process is done in three steps: Firstly, we extract necessary data from TU Wien internal databases. Then, we define a set of different deep learning architectures and train them on our data set. The presented models are inspired by and incorporate the architectures of *LSTMs*, *Autoencoder* and *Siamese Neural Networks*. Each model is trained based on three text embedding modules: *BERT*, *GPT2* and *XLNet*. Finally, we evaluate the models on the task of ranking potential research profiles given a thesis as input and compare it with BM25, an established state-of-the-art baseline. Furthermore, a manual evaluation for selected use cases is performed to further validate their respective performances. We conclude that models based on a *Siamese Neural Network* architecture achieve promising results and, in the setting of neural re-ranking, even outperform BM25. Based on the conducted experiments, we also observe that *BERT* as embedding module results in the best scores across all architectures. Finally, we come to the conclusion that the matching and selection process of reviewers can be optimised using the above presented state-of-the-art deep learning methods.

Contents

CHAPTER 1

Introduction

When it comes to the integrity and correctness of contributions in the scientific domain such as papers, the concept of peer review plays a crucial role. Based on the general topic of the respective contribution and its related fields, reviewers with profound background knowledge must be selected to ensure their ability to evaluate thoroughly. Therefore, scientific conferences, journals and other instances for publishing work follow strict guidelines paired with an extended process for conducting peer review. However, finding the best possible fit of reviewers for any given scientific work still is - in many cases - subject to a manual process, making it vulnerable for human errors and, thus, mistakes in the evaluation of the correctness and impact of the contribution.

In this context, we motivate the topic and content of this thesis as the process of graduating at TU Wien shares many similarities with the above presented problem.

1.1 Motivation

As part of a successful graduation at TU Wien for any of the available master programmes, students need to compose a master thesis with focus on parts of their studies content. Additionally, after the thesis has been submitted to the dean of student affairs, the students must defend their work in front of an examination board consisting of the supervisor and two additional reviewers. Purpose of the examination is to prove that the respective student understands and is able to apply the process of scientific research with focus on the main topic of her or his composed thesis. Depending on the topic, the evaluation of which reviewers represent the best fit for examining the student can range from being a very simple and obvious choice, to being a rather complex and tedious task, leading to a potential mismatch between reviewer and student. Any kind of mismatch in this scenario could cause a faulty evaluation of the examinee, thus, the process of selecting potential reviewers needs to be performed thoroughly. However, a perfect manual selection for

every student is not always feasible due to the high number of employed researchers and graduates at TU Wien as well as a variety of different constraints such as available time slots and the topic of the thesis.

Therefore, an automated but trustworthy solution is desired by TU Wien with the aim of optimising the matching and selection process of reviewers for a given master thesis. With current advances in the field of machine learning and the recent success of natural language processing algorithms using deep neural networks, this thesis aims to tackle the above stated problem using state-of-the-art deep learning architectures. Successful language models such as *BERT* [DCLT19] will be applied and evaluated based on our research question:

> *Can the matching and selection process of thesis reviewers be automated, based on research profiles and the text corpus of a thesis as input? Furthermore, can the quality of reviewer-thesis matches be improved using state-of-the-art methodology in the area of Artificial Intelligence?*

Furthermore, this thesis aims to contribute to the general problem statement of matching scientific content and peer reviewer with expertise in the corresponding field of research. Due to the mentioned similarities between both topics, the results of the evaluation - based on TU Wien as primary use case - suggest that similar performance might be achievable for use cases of bigger scope such as paper-reviewer-matching for conferences.

1.2 Approach

As previously described, the outcome of this thesis is based on deep learning models acting as recommendation engine to match and select potential reviewers for the committee of the final exam. The content of the examination is strongly related to the subject of the respective master thesis. The required data will be extracted from TU Wien internal systems, with additional preprocessing performed to harmonise data.

For the development of the deep learning models, we conduct a literature review of related work and state-of-the-art methodology in processing natural language. Based on this research, we introduce suitable network architectures, which will be trained and evaluated on the extracted data. For the final conclusion, all conducted experiments will be compared and measured against each other, including state-of-the-art baselines.

1.3 Outline

Following the defined approach, this thesis will be structured into four main parts. In the first part of this thesis, we introduce relevant background knowledge in the area of *Natural Language Processing* and *Information Retrieval* including related work and methodology, as this is required to follow the subsequent content and experiments.

Afterwards, we describe the underlying data set used for all of our experiments. This includes the specification of all required data sources and the respective extraction process, important pre-processing steps, as well as a detailed description of the resulting base data set.

The main part of this thesis will present all developed deep learning models and conducted experiments. As a prerequisite, we first specify the setup of the experiments and define all technical requirements. Then, the respective experiments will be described in detail, including a comprehensive overview of the corresponding evaluation results.

Finally, we conclude this thesis with a summary of our results and a detailed comparison of the model's performances. In addition to that, we also present possible areas of improvements and further ideas as part of a general outlook on future work.

CHAPTER 2

Preliminaries

As motivated in the previous chapter, the scope of this thesis revolves around building machine learning models which are able to process natural language in the form of written text. Therefore, we need to introduce a set of preliminaries required to follow and understand the content of this thesis.

We will start by presenting (1) a brief introduction to the research field of *Natural Language Processing (NLP)*, followed by a deep dive into (2) information retrieval for NLP, introducing state-of-the-art language models which are used for the experiments in Chapter 5. Subsequently, we will continue by presenting (3) related work and literature influencing the content of this thesis. Finally, we will end the chapter with introducing additionally required methodology such as (4) the comparison of document ranking and classification in the context of this thesis and (5) measures to calculate similarity between two given vectors.

2.1 Natural Language Processing (NLP)

The research field of *Artificial Intelligence* offers a wide variety of subcategories, such as *Computer Vision* for processing images, *Speech Recognition*, and many more. When it comes to natural language, we call the field of research *Natural Language Processing (NLP)*. *NLP*, as a general term, relates to the act of processing and understanding human language from a technical point of view, i.e. using computers and algorithms designed for the respective purpose. The main objective of NLP methods and algorithms lies in the ability to mimic human abilities such as speaking, sharing knowledge, writing books and reports, and understanding semantic information.

There are various kinds of techniques within the area of NLP such as statistical language modelling and parsing [RMPA16]. For the scope of this thesis, however, we will only give a very brief overview over the methodology used for *syntactic* and *semantic* analysis, and focus on explaining terms that are relevant in our context.

Syntactic Analysis. The term *Syntax* in the context of human language refers to grammar and related rules to construct grammatically correct sentences. Part of the syntactical analysis of text is the already mentioned *Parsing* method, which generates a tree presenting the different components of a given sentence (see Figure 2.1 for reference). *Part-of-Speech (POS) Tagging*, another related method, tries to identify and name each individual part of the sentence, e.g. verbs, nouns, and adjectives.

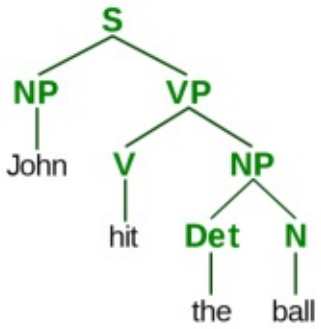

Figure 2.1: Example of a Parsing Tree [RMPA16]

Two methods, which are more relevant for data preprocessing and, thus, for the context of this thesis, are *Lemmatization* and *Stemming*. Both approaches aim to harmonise inflected words by reducing them to a shared root. The process of *Stemming* uses the related word stem as root, whereas *Lemmatization* looks for the lemma of the respective words, i.e. a root word that shares the meaning of the respective words. As an example, the stemmed form of the words *change*, *changes*, and *changing* is *chang*, the lemma is *change* [KSMM20].

Semantic Analysis. In contrast to the syntactical analysis, the term *Semantic* is strictly related to the intended meaning of words, sentences, and documents of any kind. Methods related to the semantic analysis are *Named Entity Recognition (NER)* [BRN+09], which aims to identify defined entities such as persons or locations within a text, and *Word Sense Disambiguation* [Nav09], where the algorithm assigns meaning to a word based on the given context.

Looking at applications such as document classification and ranking, however, the resulting features generated by NLP methods are often used in combination with machine

and deep learning models, which rely on the information being presented in the form of vectors. Thus, we will present additional methodology related to the field of **Information Retrieval** in the next section.

2.2 Information Retrieval

When it comes to information retrieval for natural language, i.e. retrieving and processing documents and other text related data, there are different approaches of transforming written text to data which can be understood by machine or deep learning models. Transforming text into a numeric format is required as every learning algorithm is making use of numeric computations in order to process the data accordingly. Traditional one-hot encodings for each possible word would lead to an unreasonable requirement of storage space, while also not containing any kind of context information of the document they appear in.

Established methods which make us of representing text data in vector form, while also containing contextual information, are *Term Frequency–Inverse Document Frequency (TF-IDF)* [Ram03] and *BM25* [RWJ+94]. The first step for creating the required underlying data structure is to generate a term frequency matrix. The term frequency matrix tf is of dimension $|T| \times |D|$, where T is the set of all terms occurring in all available documents and D is the set of all documents. For each term t, e.g. *language*, the respective count for each document d is stored as matrix element $tf_{t,d}$ (see Table 2.1).

	Document 1	**Document 2**	**Document 3**	**...**
language	10	0	1	...
learning	7	2	3	...
machine	5	0	0	...
natural	8	1	0	...
processing	5	0	7	...
task	2	3	1	...
...	...	...	...	...

Table 2.1: Example Term Frequency Matrix

As the resulting matrix will contain many 0-elements, the term frequencies are usually stored for non-0 entries only. The resulting data structure is called *Inverted Index*, where for each term t a pair of document ID and term frequency in the respective document is stored. The *Inverted Index* will then be used to calculate frequency scores for the respective retrieval models below.

TF-IDF. Let $tf_{t,d}$ be the *Term Frequency (TF)* for the term t in the document d, and df_t the number of documents in which the term t occurs. The *TF-IDF* scoring function is then defined as

$$TF\text{-}IDF = \sum_{t \in T_d \cap T_q} tf_{t,d} * \log(\frac{|D|}{df_t}) \tag{2.1}$$

where D is the collection of all documents and $\log(\frac{|D|}{df_t})$ the respective *Inverse Document Frequency (IDF)*. The final score is calculated as sum of all terms appearing in both, the query q and the document d, thus $\sum_{t \in T_d \cap T_q}$, where T_q and T_d are the sets of terms for the query and the document respectively [Hof19].

As individual documents within the collection D might differ in length, this might distort the corresponding term frequency values. *BM25* is another scoring model that considers the length of the respective document as well as the average document length in the complete set. Although originally introduced in the year 1994 [RWJ+94] and showing better performance than *TF-IDF* for many tasks, *BM25* was set as the default scoring function in Apache Lucene only in the year 2015 [Tur15].

BM25. Let $tf_{t,d}$ be the term frequency for the term t in the document d, and df_t the number of documents in which the term t occurs. For each document d, the respective length is represented by dl_d, with $avgdl$ being the average document length in the complete set of documents D. The *TF-IDF* scoring function is then defined as

$$BM25 = \sum_{t \in T_d \cap T_q} \frac{tf_{t,d}}{k_1((1-b) + b\frac{dl_d}{avgdl}) + tf_{t,d}} * \log \frac{|D| - df_t + 0.5}{df_t + 0.5} \tag{2.2}$$

where k_1 and b are additional hyperparameters to weight the corresponding variables. The final score is calculated for all terms appearing in the query q as well as the document d, where T_q and T_d are the sets of terms for the query and the document respectively [Hof19].

Another possibility to encode text bodies into dense vectors is to make use of neural network architectures. Variants like *Word2vec* [MSC+13] and *Glove* [PSM14] achieved promising results in various application fields of NLP, becoming a new state-of-the-art information retrieval method. With the introduction of a neural network architecture called *Transformer*, new language models have been developed, outperforming the previous approaches. For details on the architecture of the *Transformer*, we refer to the original paper [VSP+17]. In the following sections, we will introduce three different language models which will be essential to our experiments: (1) *Bidirectional Encoder Representations from Transformers (BERT)*, (2) *Generative Pre-Trained Transformer 2 (GPT2)*, and (3) *XLNet*.

2.2.1 BERT

The first language model we will present is *BERT (Bidirectional Encoder Representations from Transformers)*, which was originally introduced by the *Google AI Language* team in [DCLT19]. *BERT* is based on the encoder part of the *Transformer* architecture and uses bidirectional connections across each individual *Transformer* block. This means that each word will be enriched with context words before and after its position within the text. The architecture was inspired by *OpenAI's GPT* language model, which only uses context words before the respective word as additional information (see Figure 2.2 for reference).

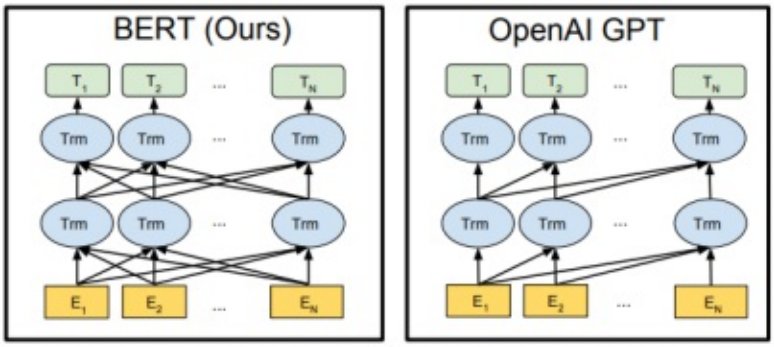

Figure 2.2: Architecture of BERT compared to GPT [DCLT19]

The language model is trained based on two unsupervised tasks: (1) *Masked Language Modelling (MLM)* and (2) *Next Sentence Prediction (NSP)*. An overview of the process can be seen in Figure 2.3.

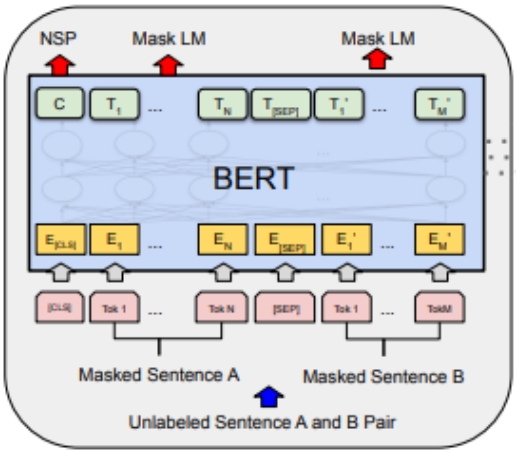

Figure 2.3: Training Procedure for BERT [DCLT19]

MLM. *Masked Language Modelling* is performed by randomly masking out a word in a given sentence: Let *"my dog is hairy"* be our sample sentence. *BERT* is then choosing a random word in the sentence and then applies the masking procedure. In 80 percent of the cases, the masking procedure replaces the word with the token $[MASK]$ (*"my dog is [MASK]"*), in another 10 percent, the word is replaced by another random word (*"my dog is apple"*). For the last 10 percent, the word will remain untouched, resulting in the same sentence as before. The task for the model during the training is to predict the correct word for all respective $[MASK]$ tokens.

NSP. The second part of the training consists of *Next Sentence Prediction*. In case of *BERT*, this is done by using pairs of two sentences A and B with a corresponding label representing whether sentence B follows sentence A. The model then has to correctly predict the respective label, given a sentence pair as input.

The resulting model can be further used for transfer learning, i.e. using the pre-trained model for tasks different to MLM and NSP. *BERT* achieves state-of-the-art performances across several different categories such as general language understanding and question answering [DCLT19].

2.2.2 GPT2

In contrast to *BERT*, the language model *GPT2*, as proposed by *OpenAI* in [RWC+19], is based on the decoder part of the *Transformer* architecture. Another difference in the models is also the learning objective: While *BERT* is trained using two different unsupervised tasks, MLM and NSP, *GPT2* has the main objective to generate language. This means that *GPT2* aims to predict subsequent words, one by one, based on the previous context. Thus, as already mentioned in Section 2.2.1, *GPT2* only uses a unidirectional architecture (see Figure 2.2 for reference). The learning objective is achieved by using *autoregressive language modelling*, i.e. the model tries to estimate the probability distribution of different text bodies. In more technical terms, this means that, given a sequence of text $s = (s_1, s_2, ..., s_n)$, the model calculates $p(s) = \prod_{i=1}^{n} p(s_i \mid s_1, ..., s_{i-1})$ [YDY+19].

In comparison to its predecessor *GPT*, the newer version *GPT2* only uses minor changes to improve. The team of *OpenAI* developed a new non-public data set, which is bigger than the one used for the previous version of the model. Additionally, the architecture of the biggest *GPT2* model has ten times more parameters, which is achieved by scaling the architecture using multiple decoder blocks [RWC+19].

As presented in [RWA+19], *GPT2* achieves state-of-the-art performance when it comes to text generation, where the model is able to write cohesive paragraphs for a suggested topic close to human quality. For non-domain-specific tasks such as question answering

and reading comprehension, the *Transformer* based network fails to perform on the level as other state-of-the-art models such as *BERT*, but achieves promising results nonetheless as they indicate the possibility for using *GPT2* for transfer learning. On an interesting note, the authors also mention that they fear malicious use of the model, which is why they refused to release the underlying data set and their biggest pre-trained model (a smaller version was released instead).

2.2.3 XLNet

Similar to *GPT2*, the language model *XLNet*, introduced by the *Google AI Brain* Team in [YDY+19], is based on the decoder part of the *Transformer* architecture. Both models also share the underlying learning objective, i.e. estimating the probability distribution of text corpora. However, *XLNet* tries to improve its performance on both, *GPT2* and *BERT*, which is done by considering bidirectional context information as in the case of BERT but at the same time avoiding its discrepancy of input data in the training and inference process caused by the random masking process.

As *XLNet* is based on a unidirectional architecture as well, the incorporation of bidirectional context information needs to be handled differently. Therefore, *Permutation Language Modelling* is introduced in [YDY+19], which uses all possible permutations of the factorisation order (see Figure 2.4 for an example).

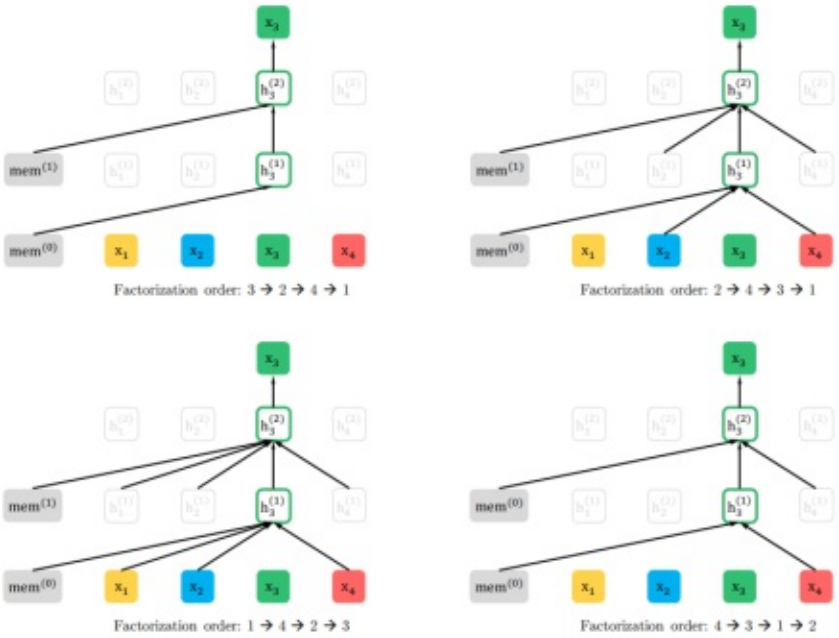

Figure 2.4: Permutation of the Factorisation in XLNet [YDY+19]

It is important to note that the input sequence will retain its ordering and only the permutation of the factorisation is considered. By integrating the benefits of autoregressive language modelling and bidirectional context information, *XLNet* manages to outperform *BERT* in several different domains of NLP such as language understanding, reading comprehension, text classification, and document ranking [YDY+19].

2.3 Related Work

Following the introduction of *Natural Language Processing* including methods and models of different kind, we now present various applications and other research related to the context of this thesis.

As the main objective of this thesis is to match potential reviewers with a given thesis, the relation to the review process of papers for conferences is obvious. In [AGB+19], the authors introduce several approaches aiming to optimise the matching process of papers and reviewers. The main concern stated in the publication is the mismatch of vocabulary used in submitted papers and the related topics of potential reviewers. This also relates to the scope of this thesis as we cannot expect perfect matches between a thesis and researcher employed at TU Wien, meaning that a researcher might be a potential fit without sharing the exact vocabulary with the given thesis as part of their scientific background. The approach, as presented in [AGB+19], is based on word embeddings generated by *Word2vec* [MSC+13] for both, paper and the background information of the reviewer. They then continue to calculate a similarity score based on the *Cosine Similarity* (see Equation 2.3 for reference) for topics extracted from both documents respectively. In addition to that, they introduce a relevance scoring to quantify how meaningful the selected common topics are for the submission, which is calculated using the previously defined similarity value. For the final *reviewer-submission relevance*, the authors use the harmonic mean of the relevance scores reviewer-submission as well as submission-submission (by comparing the content to the respective topics).

A different approach, related to the problem of matching potential reviewers with paper submissions, is presented in [ZZD+20]. Instead of using a similarity function to calculate respective scores to rank the reviewers, the authors present a solution based on multi-label classification. For each paper and profile of a reviewer, a hierarchical embedding procedure is introduced. First, sentence embeddings are generated using a bidirectional Gated Recurrent Unit (GRU) [CGCB14]. Then, an embedding for the complete text corpus is calculated with a second bidirectional GRU. Based on the generated embeddings for both, reviewer and paper submission, multi-label classification is performed to predict possible categories of the respective documents. Afterwards, potential reviewers are being matched with a given submission based on predicted labels they have in common (see Figure 2.5 for reference).

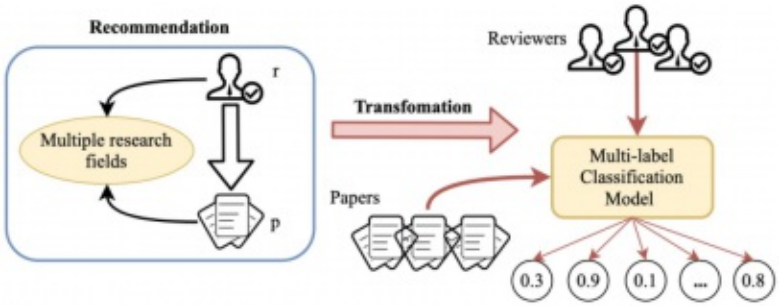

Figure 2.5: Process of Matching Reviewers with Paper Submissions [CGCB14]

Following the approach of multi-label classification, a variant of *BERT* for classifying whole documents is introduced in [ARTL19]: *DocBERT*. In this publication, the authors enhance the original architecture of *BERT* with a single fully-connected layer on top of it for multi-label classification. They then further combine the task of classification with the task of distilling knowledge (based on an LSTM) by calculating an added loss value of both models during the training process.

In [BS20], a neural network with the purpose of entity matching - i.e. identify data instances that match with real-life entities - is proposed. The model is also based on *BERT* to extract embeddings for the respective instances and entities, paired with a fully-connected layer resulting in two output neurons. Based on two given input samples, a data instance and a real-life entity, the network concatenates them and feeds them into the pre-trained *BERT* module, followed by the fully-connected layer calculating the probability of being a potential match or not. Although this approach follows the methodology of classification, it differs from the ideas presented in [ARTL19] and [ZZD+20] as it directly compares two given samples and determines a single label.

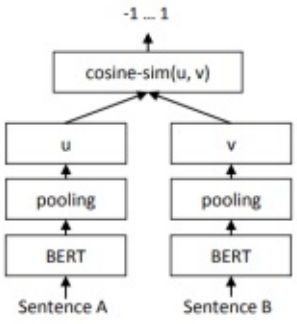

Figure 2.6: Example Architecture of Sentence-BERT [RG19]

Another variant of *BERT*, related to the approach of ranking documents rather than classifying them, is introduced as *Sentence-BERT* in [RG19]. The authors propose

a neural network architecture which takes two sentences as input and computes the respective similarity score. The architecture, as seen in Figure 2.6, is inspired by *Siamese Neural Networks* and uses *Cosine Similarity* (see Equation 2.3) to calculate the similarity score.

In [MC17], the authors present a comprehensive overview over neural models in information retrieval. In the context of *learning to rank*, *Siamese Neural Networks* are introduced as core architecture for neural ranking models, consisting of three main components: (1) generation of an embedding for the query in scope, (2) generation of an embedding for the respective document which is compared to the query, and (3) a module to match both embeddings and calculate the relevance of the document in regards to the query.

However, the idea of *Siamese Neural Networks* as architecture to calculate similarity is not specific to the domain of *Natural Language Processing*. In [Koc15], *Siamese Neural Networks* are introduced as application for *Facial Recognition*, which relates to the research field of *Computer Vision*. The motivation of using a neural network architecture that computes similarity scores for images rather than classifying them lies in the availability of data. When training the network, the data set contains a collection of images showing the faces of various people. However, when deploying a model aiming to recognise the face of a specific person, he or she might not have been represented in the training data set. Therefore, the model inspired by the concept of *Siamese Neural Networks* aims to act as a similarity scoring such that images of unseen persons can be added to the database without the need of retraining the model. As an example, we consider person P to be added to the pool of potential faces to be recognized, while not being included in the original training data set. Whenever the model gets an image of the face of person P, it will compare it with all images in the pool of potential matches including the newly added image of person P. The network will then rank the similarity scores, with the likely scenario of the image of person P having the highest similarity score. The concept of training neural network architectures which are able to correctly match or classify input data based on only a few samples is also known as *Few-Shot Learning* (or *One-Shot Learning* for a single given sample respectively).

2.4 Document Ranking versus Classification

As presented in Chapter 2.1 and 2.3, there are many different methods and applications to process text and documents. In the context of this thesis, we can narrow them down to two core ideas: (1) document ranking and (2) document classification. The motivated goal for this thesis is to match a given thesis with potential reviewers as part of the examination process, hence, we can either present a ranked list of reviewers to the user or try to use the pool of reviewers as classification target.

Document Classification. For classifying a set of given documents, the machine learning model is built based on a predefined number of possible classes, derived from the target labels of the documents in the training data set. From a technical point of view, this means that the machine learning model will output an n-dimensional vector, where n is the number of classes. The *Softmax* function is commonly used to scale the elements of the output vector such that $\sum_{i=1}^{n} x_i = 1$, with x_i being the i^{th} element of the vector, representing the corresponding class. The resulting vector elements can then be understood as a kind of confidence score for the final classification of the underlying document, where usually the element with the highest score dictates the class label. In the context of this thesis, the pool of reviewers would represent the set of possible classes and the thesis the corresponding input we want to classify.

Document Ranking. In the case of ranking documents, the machine learning model calculates a ranking score for each query-document-pair. Thus, for each query, the model generates a list of scores for the set of underlying documents, which can then be ordered in descending order to create a ranked list of documents. In contrast to the classification problem, the ranking model outputs a 1-dimensional vector representing the ranking score regardless of the number of initially available reviewers. In our context, a given thesis represents the query and the pool of reviewers the set of documents which has to be ranked.

Based on the above presented aspects related to classification and ranking, we decide to follow the approach of *Document Ranking* for the scope of this thesis. The main reason behind this decision is that the pool of potential reviewers might be fixed for our experiments (see Chapter 5 for details), in the context of the future application of our machine learning model, however, additional researchers could be added as potential reviewer. Following the approach of *Document Classification*, we are only given a static set of possible classes (or reviewers in this context) for our model, i.e. a given thesis can only be matched with the initial reviewers our model was trained with. By using a model which ranks the reviewers based on a given thesis, we hope that our neural network learns to generalise well enough to generate meaningful ranking scores for previously unseen reviewers. In other words, no re-training of the final machine learning model is required when extending the original pool of potential reviewers.

2.5 Similarity Measures

In order to generate a ranked list of potential reviewers, we have to define a function that measures the similarity of given thesis-reviewer-pairs. The intention of using similarity between two documents as core measurement for the ranking process is the assumption that embeddings of reviewer profiles closer to the embedding of a given thesis represent a better match than those with a bigger distance (the term *distance* is used as opposite to the term *similarity*). As introduced in Chapter 2.2, we can generate text embeddings in the form of vectors, thus, we can use common distance and similarity metrics for vectors

such as the *Cosine Similarity* [VK16].

Cosine Similarity. Let A and B be two given vectors, where a_i and b_i are the i^{th} element of each respective vector and $|A| = |B|$. The *Cosine Similarity* function is then defined as

$$Cosine\ Similarity = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}. \tag{2.3}$$

with n representing the total number of elements in each given vector.

However, as the *Cosine Similarity* calculates a similarity score in the range $[-1, 1]$ (where -1 and 1 represent maximal dissimilarity and similarity respectively), we follow the suggestion presented in [CYK+18] and use the *Angular Similarity* function as measurement to generate similarity scores in the range $[0, 1]$.

Angular Similarity. The *Angular Similarity* for two given vectors is defined as

$$Angular\ Similarity = 1 - Angular\ Distance \tag{2.4}$$

where the *Angular Distance* metric is calculated based on the respective *Cosine Similarity* between two given vectors:

$$Angular\ Distance = \frac{cos^{-1}(Cosine\ Similarity)}{\pi}. \tag{2.5}$$

Using *Angular Similarity* instead of *Cosine Similarity* for calculating the similarity between two given vectors introduces an additional benefit for our application. In Figure 2.7, we present a comparison of both functions, where the *Cosine Similarity* function is scaled to the range $[0, 1]$ accordingly. We can see that the *Angular Similarity* function is not strictly linear, meaning that high similarity or dissimilarity are separated by a bigger margin.

In addition to *Angular Similarity*, we will introduce a similarity function based on the *Manhattan Distance* metric [Thy15]. Similar to the motivation of using *Angular Similarity*, we will take the exponential negative distance to create a stronger separation of similar and dissimilar documents. Using the negative distance value instead of the original one is important as we want the similarity function to calculate scores in the range $[0, 1]$, with 0 and 1 representing dissimilarity and similarity respectively.

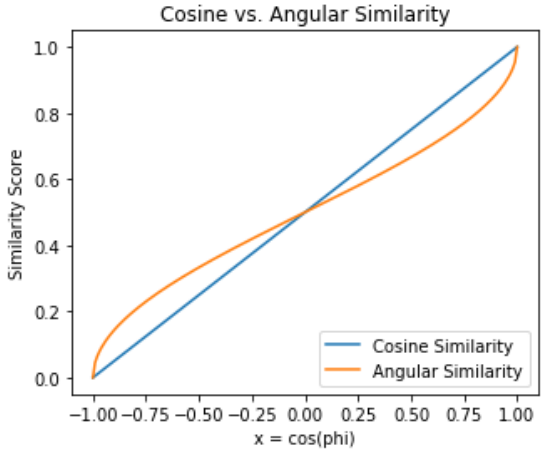

Figure 2.7: Comparison of Cosine Similarity and Angular Similarity

Manhattan Similarity. Let A and B be two given vectors, where a_i and b_i are the i^{th} elements of each respective vector and $|A| = |B|$. The *Manhattan Similarity* function is then defined as

$$Manhattan\ Similarity = \exp(-\sum_{i=1}^{n} |a_i - b_i|) \tag{2.6}$$

where n is the total number of elements in each given vector.

In contrast to the *Cosine Similarity* function, the *Manhattan Similarity* is based on the distance between all elements of two given vectors rather than their respective angle. For example, two vectors with equal angles but differently scaled values will result in a similarity of 1.0 when using the *Cosine Similarity*, but - based on the size of the vector - yield a rather low score for the *Manhattan Similarity*.

After introducing the most important preliminaries in the context of this thesis, the following chapters will present additional requirements needed for conducting our experiments.

CHAPTER 3

Data

The first step towards building a machine learning model is defining the data set which will be used to train the respective model. Thus, this chapter will introduce the data set which will be used throughout all following chapters. As we are not relying on a publicly available data set, motivated by the scope and the problem description of this thesis, we will present the complete process from gathering the required data to the final base data set for our experiments.

First, we will (1) define all source systems required to collect the needed data, followed by (2) the pipeline and necessary supporting software in order to extract it accordingly. Afterwards, we will provide a brief but comprehensive analysis of the extracted documents, leading to (3) the description of important preprocessing steps. In the last section of this chapter, we will (4) introduce the base data set, including the definition of the ground truth and additional methodology required for the different experiments.

3.1 Sources

As motivated in the beginning of this thesis, we want to rank research profiles of potential reviewers based on the content of any given thesis. Therefore, we need to extract two different types of documents: theses and research profiles.

Research Profiles. In order to extract research profiles of potential reviewers, we will refer to internal services of TU Wien: *TU Information Systems and Services (TISS)*[1]. TISS provides an API and a comprehensive documentation[2] of different API calls, which also include the possibility to search for the profile of an employee. As this profile includes

[1] `https://tiss.tuwien.ac.at`, last visited on 10-01-2021

[2] `https://tiss.tuwien.ac.at/api/dokumentation`, last visited on 10-01-2021

a section for professional background and research topics, we will use the API of TISS to retrieve the required research profiles.

Theses. For the past years, all submitted theses at *TU Wien* have been stored and uploaded to the online repository *CatalogPlus*[3], including meta information such as author, year, and abstract as well as the PDF of the original thesis. To collect semantic information of a given thesis, the obvious choice would be the complete text body of the written document. For the scope of our experiments, however, we will extract only the abstract from the presented repository. There are two main reasons for this decision: First, An abstract briefly describes the work presented in a given thesis, including the general topic as well as methodology. Therefore, the semantic information of the abstract should be sufficient to determine whether a specific research profile is a relevant match or not. The second reason is the disproportionately higher effort that is required to extract textual content from a PDF, compared to the additional amount of semantic information it would provide.

3.2 Pipeline

Following the definition of the data sources, we continue by specifying the data pipeline, i.e. the process of extracting the required data from its source systems. As we are using two different document types - abstracts and research profiles - with different sources, this section will be further split into a subsection for *Research Profiles* and *Abstracts*. For the implementation of the data pipeline, the programming language *Python* [VRD09] will be used.

3.2.1 Research Profiles

Following the definition of the source system for research profiles, we will use a sequence of different API calls to extract information about potential reviewers from TISS, including their respective research profile. First, we start by extracting all research units at TU Wien[4]. By parsing the result of the API call, we extract the *ID*, *Name*, *Code*, and *Parent_Unit* for each research unit.

In the next step, we iterate over all research units and use their respective *Code* as input for the API call[5]. This will result in a list of all employees associated with the research unit, for which we store the attributes *First_Name*, *Last_Name*, *Preceding_Titles*, *Postpositioned_Titles*, *Function*, *Research_Unit*, *ID*, and *OID*. In case no *ID* is assigned to the employee, the entry will be populated with a unique default value.

[3] `https://catalogplus.tuwien.at`, last visited on 10-01-2021
[4] `https://tiss.tuwien.ac.at/api/fpf/rest/research_unit/get_all`
[5] `https://tiss.tuwien.ac.at/api/orgunit/v22/code/<UnitCode>?persons=true`

Then, for each employee, we process a our next API call[6] based on her or his *ID* to retrieve the corresponding employee profile. The resulting XML document will be further parsed to collect all relevant background information in English.

In addition to the complete list of research profiles, a separate subset will be generated, including only employees which are eligible for the review process of theses. An employee is eligible if she or he possesses one or more of the following titles: univ.prof, assoc.prof or habil. This collection of profiles represents the final pool of potential reviewers as required for the final evaluation of our experiments (see Chapter 6 for details).

3.2.2 Abstracts

The defined source system for retrieving the abstracts, *CatalogPlus*, does not provide an API. Therefore, we use an additional Python library called *Beautiful Soup* [Ric07], which will allow us to scrape content from websites.

Based on the retrieved research profiles, we will start collecting abstracts by looping over all reviewers. For each reviewer, a search request is sent to *CatalogPlus*, using the respective first and last name of the reviewer as additional search criteria. The resulting website lists all theses related to the reviewer, i.e. all theses she or he supervised in the past. However, as the results are limited to 20 entries per page, we use *Beautiful Soup* to retrieve the URLs of all subsequent result pages, storing them in a separate list. Then, for each individual result page, we will extract the URLs of all websites linking to individual theses and their abstracts.

After storing the URLs of all results, the content of each of the websites will be retrieved using the built-in HTML-parser of *Beautiful Soup*. The resulting text body contains various information related to the respective thesis such as author or year, but also the German and English version of its abstract. Thus, we will further process the content to extract the English version of the abstract and store it accordingly with a reference to the original supervisor.

3.3 Preprocessing

In order to use the raw data (retrieved from the presented data pipeline) for the creation of the base data set, the format and quality of the documents need to be harmonised. This section will describe all preprocessing steps for both, (1) research profiles and (2) abstracts.

[6] `https://tiss.tuwien.ac.at/api/fpf/rest/research_person/profile/<ID>`

3.3.1 Research Profiles

As the research profiles are parsed from an XML file, there are various elements within the text which do not represent semantic information, e.g. XML tags or timestamps (see Listing 3.1). Therefore, the first part of the preprocessing of the profiles consists of the removal of timestamps in the format of *yyyy-mm-dd hh:mm*, any special characters, and XML tags of any kind. By inspecting some randomly selected samples, we can observe that there are several terms included in the documents which do not represent valuable information, e.g. *editor*, *other*, *Patent number*, or *Beyond TUV-research foci*. Thus, more than 40 additional different terms will be removed from the content of the data set.

```
...
<ns2:personData>
...
<ns2:feiProfile>
<de>- Stereoselektive Synthese - Metabolitensynthese [...]</de>
<en>- Stereoselective synthesis - Synthesizing metabolites [...]</en>
...
<ns2:keywords>
<keyword>
<value>
<de>Asymmetrische Synthese</de>
<en>asymmetric synthesis</en>
</value>
</keyword>
...
<ns2:researchField>
<mainResearch>
<de>Ausserhalb der TUW-Forschungsschwerpunkte</de>
<en>Beyond TUV-research foci</en>
</mainResearch>
<name>
<de>Ausserhalb der TUW-Forschungsschwerpunkte</de>
<en>Beyond TUV-research foci</en>
</name>
<info>
<de>Ausserhalb der TUW-Forschungsschwerpunkte</de>
<en>Beyond TUV-research foci</en>
</info>
...
<name>
<de>SCI (Science Citation Index)</de>
<en>SCI (Science Citation Index)</en>
</name>
</classification>
<period>
<startDate>2009-09-01+02:00</startDate>
<endDate>2014-08-01+02:00</endDate>
</period>
...
```

Listing 3.1: Extract of a random Profile in XML Format

In addition to the preprocessing of unwanted symbols and terms, some of the research profiles include German content, which might have been caused by wrong user input. To harmonise the language of all research profiles, we use the Google Translation API of the Python library *googletrans* [Han18] to translate all non-English content into English.

Finally, as most of the profiles are represented by bullet points and keywords, we split the documents into independent chunks of information based on line breaks and store them as a list of character strings for each research profile.

3.3.2 Abstracts

As defined in Chapter 3.2.1, each abstract will be retrieved in English in the format of a single character string. The online repository *CatalogPlus* seems to provide already harmonised data formats for the abstracts, therefore, only one preprocessing step must be performed in the scope of this thesis: In order to process the raw text body using state-of-the-art language models such as BERT [DCLT19], we split the documents into their individual sentences and represent each abstract as list of character strings.

3.4 Base Data Set

After all required data was extracted and preprocessed, we can specify the base data set which will be used as input for all experiments. Therefore, this section will present the content and structure of the base data set, including the definition of its ground truth and supporting methodology.

3.4.1 Ground Truth

An important part of a data set well suited for machine learning experiments is the ground truth of its data samples. The ground truth is represented by the target value of each individual sample, i.e. in the case of classifying images, the respective ground truth are the class labels such as *bird* or *horse* for the domain of animals.

In the scope of this thesis, where we want to match two documents, we need to define pairs of documents that represent a good match. Following the procedure of retrieving abstracts and research profiles (see Chapter 3.2), we already store pairs of related documents, i.e. an abstract together with the research profile of its original supervisor. However, as seen in the analysis of the documents (see Chapter 3.3), not every employee provides a well documented research profile or simply no profile at all. Additionally, we cannot be certain that, in all cases, the supervisor is indeed a good match for the given thesis.

There are other methods such as clustering or topic modelling [CDOK20], which could be used to establish a ground truth. These methods, however, also strongly depend on the

quality of our input documents and might produce an unwanted bias, i.e. the clustering algorithm might exploit patterns caused by poor data quality. This will then further transfer into any model as the learning process optimizes its parameters based on the target values.

Considering all of the above aspects, we decide that the original supervisor of each abstract is the best fitting target value, resulting in the final ground truth for our base data set. To mitigate some of the impact that research profiles with poor data quality might have, we will not consider any abstracts where the profile of its supervisor was not populated.

3.4.2 Structure of the Data Set

Following the definition of our ground truth, we can now specify the final format and structure of our base data set. The data set itself is split into two different sets: (1) the set containing all abstracts including the respective target in form of supervisor *ID* together with his or her research profile, and (2) the list of all research profiles also including the respective *ID* of the employee.

The reason for this split is based on the difference in the training and evaluation process (for details see Chapter 4.4.4 and 4.4.5). The evaluation - and also the functionality of the final model - will be based on the list of potential reviewers, which is directly derived from the set of all research profiles. To avoid unnecessary computational overhead when using the final model to generate a ranked list of research profiles, they will be stored separately.

Both, the data set containing all abstracts and the data set containing all research profiles, can be further filtered to only include abstracts and profiles related to potential reviewers (see Chapter 3.2.1 for the criteria of being a potential reviewer). An overview of the final data sets can be seen in Table 3.1.

Data Set	**# of Samples**
All available Abstracts	11,198
Thereof related to Reviewers only	7,906
All available Research Profiles	1,259
Thereof related to Reviewers only	488

Table 3.1: Overview of Data Sets

Additionally, in Figure 3.1, we can see the distribution plots for both, the data based on all research profiles and the data based on only reviewer profiles. We can observe that for more than 350 reviewers in the complete data set and more than 30 reviewers in the

reviewer-only data we were not able to retrieve any related abstracts. This might impact the performance of our experiments as the affected employees will not be represented by any data samples contained in the training data set.

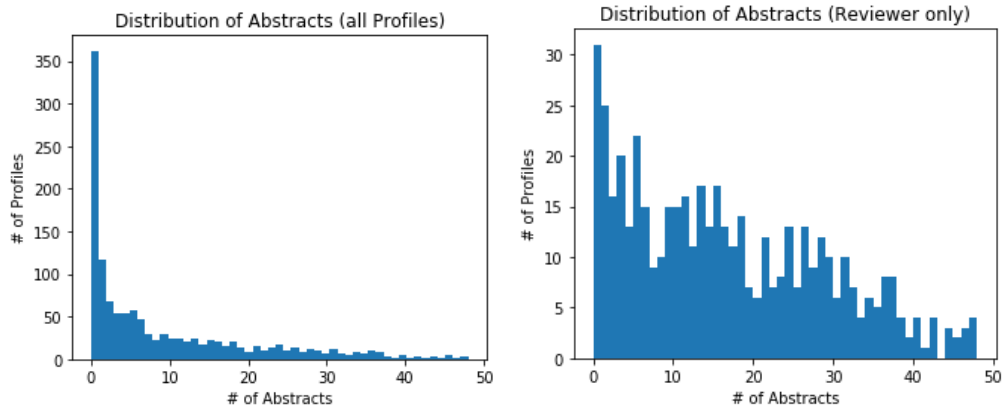

Figure 3.1: Data Distribution

3.4.3 Negative Sampling

For some of the experiments presented in Chapter 5, we need to be able to generate pairs of abstract and research profile which represent a negative sample, i.e. the corresponding reviewer is not a good match for the respective abstract. Therefore, we introduce a method for negative sampling based on a given abstract and its original supervisor.

First, we need to revisit the creation of the set containing all research profiles. The initial process for extracting the profiles (see Chapter 3.2.1 for reference) starts by iterating over all extracted research units. This means that the profiles included in the corresponding data set are ordered by research unit. The custom method *get_negative_sample* (as presented in Listing 3.2) will then take the list of research profiles (parameter *profiles*), the corresponding list of employee IDs (parameter *labels*), and the target value of the respective abstract represented by the ID of its supervisor (paramter *target*). The algorithm will then retrieve the position of the target value within the set of employee IDs and calculate a random distance value with a minimum of $\frac{n}{4}$, where n is the total number of employee IDs in the set. Based on this distance value, a new index will be calculated and the corresponding research profile returned. Due to the profiles being ordered by research unit, the defined distance will ensure that a profile of an employee belonging to a different faculty will returned. We further assume, by changing the target to a new faculty, that the new research profile will represent a bad match for the given abstract.

```python
import random

def get_negative_sample(profiles, labels, target):
    # get index of target
    index = labels.index(target)

    # calculate distance for new index
    c = len(profiles) // 4
    r = random.randint(0, len(profiles) // 2)

    # calculate new index
    new_index = (index + c + r) % len(profiles)

    return profiles[new_index]
```

Listing 3.2: Algorithm for retrieving negative Samples

Based on the introduction of the base data set and related methodology such as the negative sampling algorithm, we will continue by describing the experiment setup. The definition of the setup and related components is mandatory as it presents technical requirements and the structure needed for our experiments.

CHAPTER 4

Experiment Setup

In this chapter, we will provide an overview of the setup required to run each individual experiment. Thus, we will present a brief overview of (1) the hardware, which was used throughout all experiments, (2) the technology stack including all external libraries and frameworks, and (3) important remarks on reproducibility. In addition to those topics, we want to introduce (4) the approach used for conducting the experiments and present preliminaries for (5) the training and (6) evaluation process.

4.1 Hardware

All experiments are conducted on a single desktop computer. The respective hardware specifications can be found in Table 4.1.

CPU	Intel Core i7-7700 (3.6 GHz)
RAM	16 GB
Storage	1 TB HDD
GPU	NVIDIA GeForce GTX 1070
OS	Windows 10 Home (64-bit)

Table 4.1: Hardware Specifications used for Experiments

Due to the ability of *Graphics Processing Units (GPU)* to process vector and matrix operations in a more efficient way compared to *Central Processing Units (CPU)*, they are commonly used in the field of deep learning to reduce the training time of neural networks and speed up predictions on unseen data.

4.2 Technology Stack

For the implementation of the experiments, several libraries and frameworks covering different required algorithms and machine learning methods. In Table 4.2, we can see a list of all libraries and frameworks which were used to preprocess data, implement neural network architectures, or define custom methods and functions.

Python 3.7 [VRD09]	main programming language
NumPy [RMvdW+20]	random number generator, array & list operations
Matplotlib [Hun07]	visualisation of the training and evaulation process (e.g. loss curve, evaluation)
Scikit-learn [PVG+11]	split of data set into training, validation and test set
NLTK [BKL09]	preprocessing of research profiles (e.g. stop word removal, word stemming)
Rank-BM25 [Bro20]	implementation of BM25 and variants
HuggingFace [WDS+20]	pre-trained models for BERT, GPT2 and XL-Net including tokenizer and further preprocessing functionality
PyTorch [PGM+19]	core framework for implementing neural network architectures, defining training and evaluation methods, and creation of tensors based on the given data sets

Table 4.2: Libraries & Frameworks used for Experiments

4.3 Reproducibility

Prior to describing the general approach for the experiments and details on training and evaluation, it is important to note that any scientific experiment should be reproducible - given that the environment and related settings are the same - as this is the only possibility for reviewers and other readers to verify the correctness of the presented results. In this context, we want to point out that randomness plays a crucial role in the field of machine learning:

Data Set Preparation. Data sets are usually split into several subsets for training, validation and evaluation, which in many cases is based on a random permutation on the original data set. Depending on the samples contained in the training set and the

chosen machine learning algorithm, the performance of the developed models on unseen data can vary heavily. As defined in the previous section, *Scikit-learn* will be used as library for splitting the data set, where we can set a random seed for the parameter **random_seed** accordingly.

Initialisation of Deep Neural Networks. Based on the chosen deep learning architecture, the resulting model will contain one or more weight matrices which are used for the propagation of input vectors throughout the network. The initial value for each element in the respective matrix will be calculated at random, following a predefined initialisation method (e.g. Gaussian, Glorot). As we are using *PyTorch* as framework to implement deep neural networks, we can set a random seed for the global parameter **torch.manual_seed**.

Random Number Generator. In Chapter 3.5.2, we introduced an algorithm to retrieve negative samples, which makes use of the random number generator built into the library *NumPy*. Negative sampling is an important tool in the scope of this thesis to populate the data set with additionally needed information. Therefore, we ensure the reproducibility of the results by setting the random seed for the global parameter **numpy.random.seed**.

4.4 Approach

In order to create comparable results when conducting the experiments, we define a general approach for the structure of all experiments. This will also establish an easy to follow structure for describing the individual models in Chapter 5. Each experiment follows the same approach, which consists of (1) defining the underlying learning objective, (2) preparing the data set accordingly, (3) implementing the chosen architecture and initialising the model, (4) training the model, and (5) evaluating its performance.

4.4.1 Learning Objective

Different types of machine learning algorithms follow different objectives, e.g. classification, regression, or sequence prediction. Therefore, it is important to define the main learning objective of the respective model as this will dictate the preparation of the data set as well as the required methodology for training and evaluation.

4.4.2 Data Set

Following the defined learning objective, the data set must then be populated with required data samples and a target value for each individual sample. Without a correctly specified target for the data points, the neural network will not be able to learn meaningful patterns - and without a diverse representation of data samples, the network might inherit unwanted bias after its training.

4.4.3 Architecture

In this step of the overall approach, we specify the initial idea of the experiment and define the neural network. This includes structure and type of all network layers as well as their corresponding parameters such as number of input and output neurons.

4.4.4 Training Process

After the preparation of the required data set and the initialisation of the chosen neural network, the training process of the model can be started. Each model will be trained over a duration of 50 epochs, using *Adadelta* [Zei12] as optimiser. The training process will be monitored by calculating the average loss for both, training and validation data set, prior to the first epoch as well as after every epoch has finished.

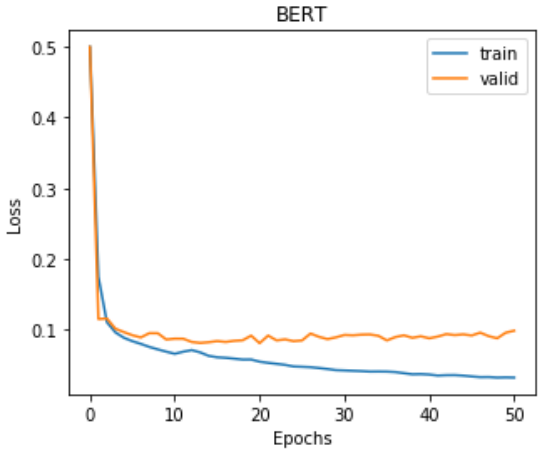

Figure 4.1: Example Plot for the Evaluation of the Training Process

In Figure 4.1, we can see an example plot showing the number of epochs on the x-axis and the corresponding loss value on the y-axis. For each individual model, a plot containing the loss curve for the training and validation data set will be generated. Additionally, for each experiment, a separate plot comparing all different models based on their training and validation error, respectively, will be used to determine the best performing model. The loss at each epoch will be calculated using either *Mean Squared Error* [SW10] or *Binary Cross Entropy*, depending on the defined learning objective of the model. The definition for both loss metrics can be found in Equation 4.1 and Equation 4.2, respectively.

Mean Squared Error (MSE). Let n be the number of scalar values in a given output vector, $\hat{y}_i$ the predicted scalar at position i and y_i the corresponding target value. The *Mean Squared Error* is then defined as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2. \tag{4.1}$$

The *Mean Squared Error* loss function is typically used for learning objectives related to regression tasks, thus, the calculation is based on the squared difference of two vectors as we expect positive and negative values. For classification tasks, we will measure the loss by taking the predicted probability of wrongly classified data samples. Therefore, we no longer need to square the difference as we only expect positive values. However, we need to differentiate between wrongly and correctly classified data samples. A loss function commonly used for the given context is the *Binary Cross Entropy*.

Binary Cross Entropy (BCE). Let n be the number of scalar values in a given output vector, $\hat{y}_i$ the predicted scalar at position i and y_i the corresponding target value. The *Binary Cross Entropy* is then defined as

$$BCE = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \; log \; \hat{y}_i + (1 - y_i) \; log \; (1 - \hat{y}_i)\right]. \tag{4.2}$$

4.4.5 Evaluation

Subsequent to the training of all individual models in a given experiment, each model will be evaluated on the problem description of this thesis: ranking research profiles for a specified abstract. Based on the definition of our base data set (see Chapter 3.5 for reference), each abstract is connected to the research profile of the original supervisor, which will serve as target for the evaluation process.

For each rank n, all correct matches will be divided by the total number of abstracts, returning a value referred to as *Accuracy* in the range $[0, 1]$. A correct match at rank n is achieved when the target research profile of the respective abstract is included within the top n results of the ranking. Therefore, for a given rank n, an accuracy value of 0.7 represents that for 70 percent of the abstracts, the target research profile is included within the first n ranked profiles. The resulting plot (as seen in Figure 4.2) then shows the performance of the respective model measured by the accuracy at each rank as well as the calculated *Mean Reciprocal Rank* [Cra09].

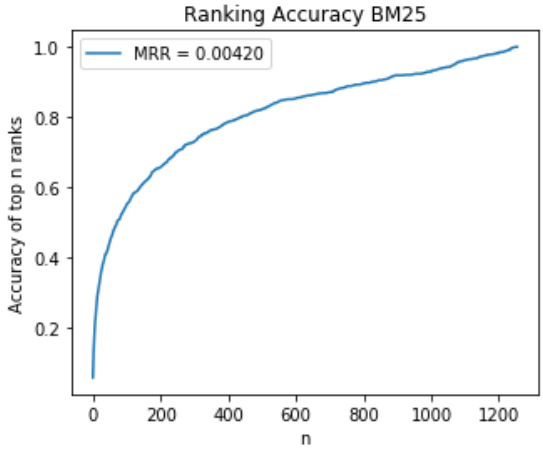

Figure 4.2: Example Plot for the Evaluation of a trained Model

Mean Reciprocal Rank (MRR). Let $|A|$ be the total number of abstracts and $rank_i$ the position of the target research profile of the i^{th} abstract. The *Mean Reciprocal Rank* is then defined as

$$MRR = \frac{1}{|A|} \sum_{i=1}^{|A|} \frac{1}{rank_i}. \tag{4.3}$$

As an example: Consider the set of abstracts $A = \{a_1, a_2, a_3\}$ and the respective target research profiles $P = \{p_1, p_2, p_3\}$. We now assume that p_1, p_2 and p_3 achieve the ranks 5, 10 and 25 respectively for the corresponding documents. The *Mean Reciprocal Rank* will then be calculated by

$$MRR = \frac{1}{|A|} \sum_{i=1}^{|A|} \frac{1}{rank_i} = \frac{1}{3} \left(\frac{1}{5} + \frac{1}{10} + \frac{1}{25} \right) \approx 0.11333 \tag{4.4}$$

With the definition of all required technical components and the overall approach, we will continue by providing a comprehensive overview of all experiments.

CHAPTER 5

Experiments

In this chapter, we will present the conducted experiments, following the methodology and approach as established in the previous chapter. First, we will set two baseline models (random scoring and BM25) which will be taken as primary source of comparison for each individual model. Subsequently, all experiments will be explained in detail including (1) the main learning objective of the respective model, (2) the underlying data set used for the training, (3) an evaluation of the training process, as well as (4) the main evaluation based on the defined metrics in Chapter 4.4.5. An overview of all models can be found below in Table 5.1.

Baselines	**Embeddings**	**Similarity Networks**
Random Scoring	Base Embeddings	LSTM-based Dual Encoder
BM25	Long Short-Term Memory (LSTM)	Siamese Angular LSTM
	Fully-Connected Autoencoder	Siamese Manhatten LSTM
	LSTM-based Autoencoder	

Table 5.1: Overview of conducted Experiments

Finally, we will end the chapter with concluding remarks on the overall performance throughout all experiments and a detailed comparison of the individual models.

5.1 Baselines

Establishing a state-of-the-art baseline is important for any experiments related to Machine Learning, especially when customized architectures are being used for the desired learning outcome. Newly trained models need to be evaluated in order to measure their

performance, but it is also crucial to measure the performance against one or more methods which have already shown great success on similar tasks, as the main objective is to outperform existing results.

For this reason, we will use two baseline models for our experiments: (1) a randomized ranking function and (2) BM25 as state-of-the-art model. In the next subsections, each baseline will be explained shortly and their respective performance will be discussed.

5.1.1 Random Ranking

A random model is often used as baseline for Machine Learning experiments as it represents a lower threshold every implemented architecture should exceed to be considered useful.

In the scope of this thesis, the random baseline is a function which takes the complete list of available research profiles as input. The abstract is missing from the required input parameters as the random scoring function follows no specific logic related to the content of the abstract other than randomly re-ordering the list of profiles. For each possible research profile, a random score between 0 and 1 is being calculated and assigned accordingly. Subsequently, the list will be sorted in descending order, i.e. rank 1 has the highest score and the last rank the lowest. As this function does not require any training or fitting to data, no train-test split is required. However, as we want to measure other experiments against this baseline, the data set is split into a training and a test set using an 80-20 ratio.

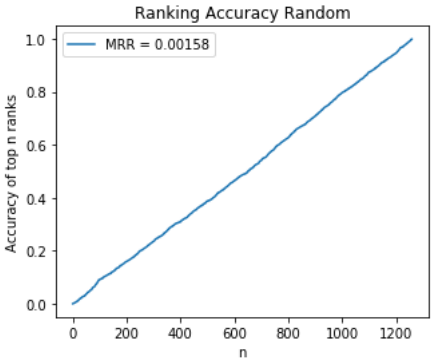

Figure 5.1: Random Scoring Baseline

In Figure 5.1, we can observe the performance of the random scoring model for the available test set of abstracts, taking the complete set of relevant research profiles as input. As to be expected, the resulting graph shows a function stretching diagonally across the plot and can be considered linear. As already mentioned in the previous

section, this model will be used as lower bound for all experiments and referred to as *Baseline Random*.

5.1.2 BM25

The second important baseline for our experiments will be set using BM25 as state-of-the-art model for several problems related to information retrieval from text and NLP in general (as described in Chapter 2.2).

In order to create the BM25 model, the underlying text corpus of available abstracts has to be defined first. Therefore, the training data set consisting of the original abstracts will be used. Each document The split into a training and a test set follows the same 80-20 ratio as used for the random model and all other experiments, with a predefined random seed to ensure that no samples will be mixed between training and test data set (see Chapter 4.3 for more details on the random seed).

The preprocessing for all documents consists of three steps: (1) lower casing of all words, (2) removing stop words, and (3) stemming. After fitting a model onto the text corpus, it can be used as a function taking the desired abstract and the complete list of available research profiles as input. It will then calculate the respective score for each profile and return the ranked list as output.

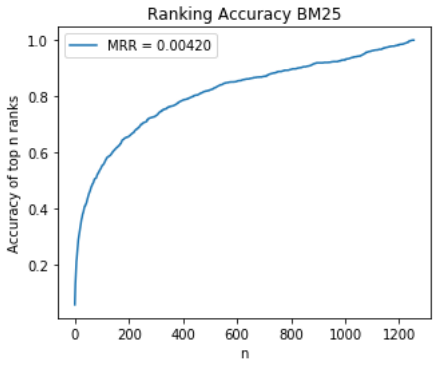

Figure 5.2: BM25 Baseline

In Figure 5.2, we can see the performance of the BM25 model on the test set, showing a clear improvement compared to the random scoring model. Due to the defined ground truth and its corresponding noisiness (see Chapter 3.5.1 for more details), however, the model still performs poorly for later ranks. This baseline will be used together with the random scoring model as baseline and referred to as *Baseline BM25*.

5.2 Base Embeddings

The first experiment will be based on the different embedding modules used in later experiments, but without any further processing of the resulting feature vectors. The three different embedding modules are BERT [DCLT19], GPT2 [RWC+19], and XLNet [YDY+19].

We will measure the performance by calculating the feature vectors for abstracts and research profiles respectively and calculating a similarity score between each abstract-profile-pair using our *Angular Similarity* function. Each embedding module will be created based on the pre-trained model available within the HuggingFace framework [WDS+20] and takes the untouched abstract or research profile as input. The model will then return a feature vector for every sentence in the respective document, which will be transformed into a single document embedding of similar dimensionality by averaging across all sentence vectors.

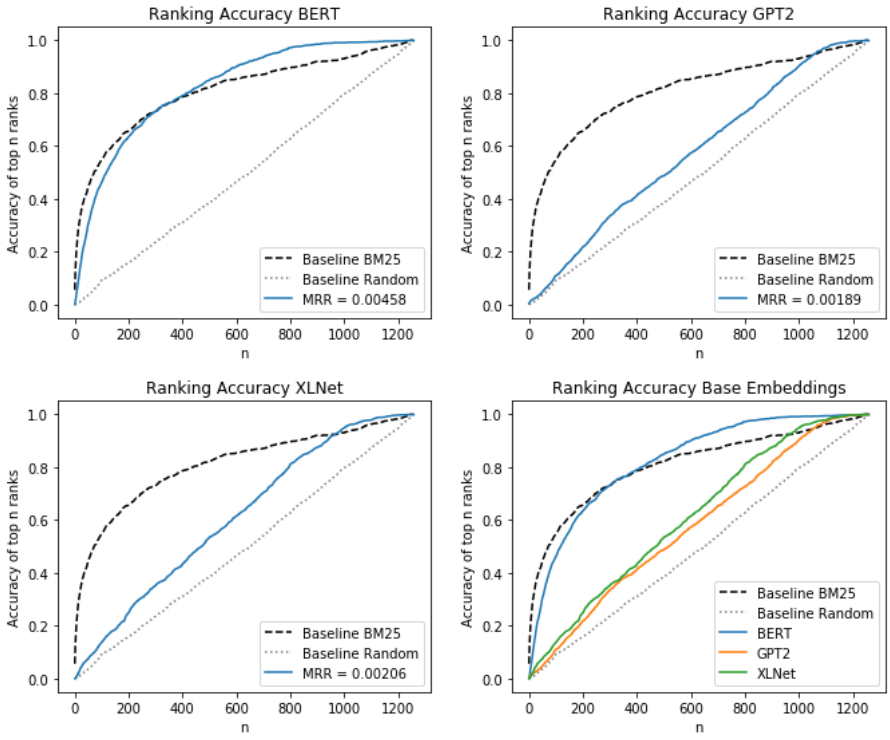

Figure 5.3: Performance of different Embedding Modules

After calculating the similarity scores for each abstract, the resulting list will be sorted in descending order and returned accordingly. As seen in Figure 5.3, GPT2 and XLNet show similar performance to the random baseline, while BM25 still significantly outperforms

them at all ranks. BERT, on the other hand, achieves a higher MRR than BM25. However, the BM25 baseline model performs better at earlier ranks, which is crucial for the scope of this thesis as the end user will be looking at the top ranked profiles only.

5.3 Long Short-Term Memory (LSTM)

As the different embedding modules generate feature vectors for each sentence of an individual document, the resulting output for each document will be a sequence of vectors. A very common network architecture for processing variable sized sequences as input is the *Long Short-Term Memory* (LSTM) introduced in [HS97]. The LSTM (and Recurrent Neural Networks in general) can be used for various tasks as seen in Figure 5.4.

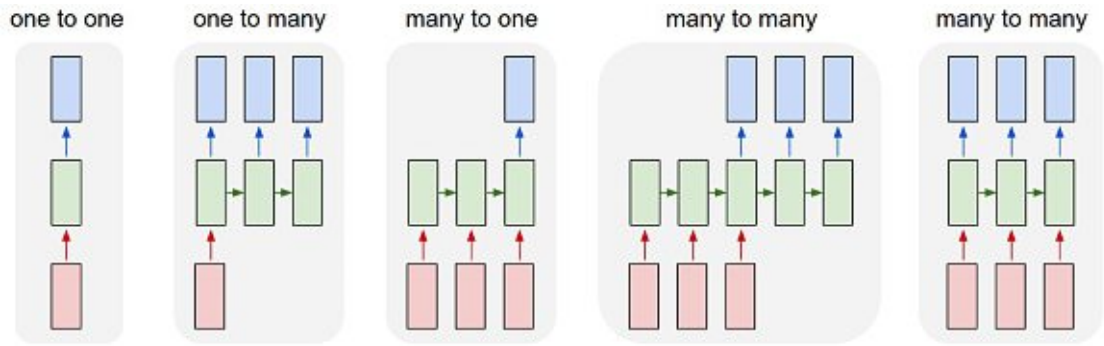

Figure 5.4: Different Ways of processing Sequences with Recurrent Neural Networks [Kar15]

For the intended idea of calculating similarity scores between two vectors for an abstract and research profile respectively, we want to generate custom embeddings based on the input sequence of sentence vectors.

5.3.1 Learning Objective

In order to learn custom embeddings which inherit semantic information of the document, the idea is to use a many-to-many LSTM architecture to predict the feature vector of the subsequent sentence based on the sentence given as input.

Therefore, the input will be the sequence of sentence embeddings s_1 to s_{n-1}, where s_i is the feature vector for the i^{th} sentence and n the total number of sentences in the document. The target sequence will then reach from s_2 to s_n, i.e. the network will be trained to predict s_{i+1} for the given input s_i.

5.3.2 Data Set

As described in the previous section, our LSTM model now takes the sequence of sentence embeddings as input and outputs another sequence of sentence embeddings. Therefore, the base data set has to be adapted in order to fit the network's purpose.

First, the part of the data set consisting of all relevant research profiles can be disregarded as the model does not need them as input. Next, for each abstract, the list of sentences will be transformed into two different lists: (1) a list containing sentences s_1 to s_{n-1} acting as input sequence, and (2) a list containing sentences s_2 to s_n for the target sequence which is required to calculate the error between prediction and target at each sequence step.

5.3.3 Architecture

The final model consists of multiple components: (1) the embedding module which is generating the feature vectors for each sentence, (2) a recurrent layer consisting of two stacked LSTMs, and (3) two fully-connected layers to transform the hidden representation into the desired output vector.

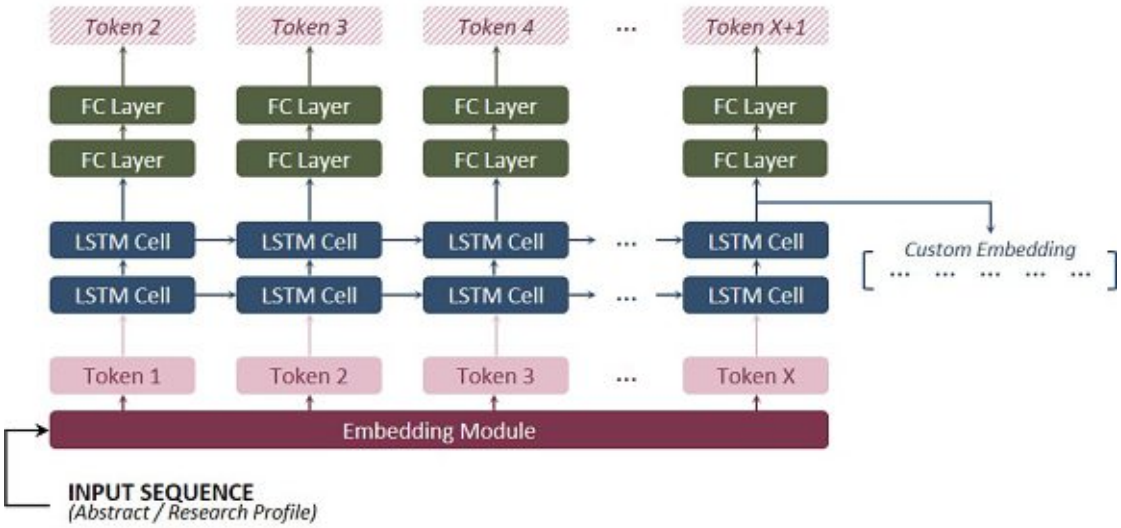

Figure 5.5: Architecture of Many-to-Many LSTM

Embedding Module. The embedding module is based on either BERT, GPT2 or XLNet - including the built-in functionality for preprocessing - and takes the respective document as input. For each sentence, a **768**-dimensional (768d) feature vector will be calculated as output (referred to as *Token* in Figure 5.5).

Recurrent Layer. The respective sentence embedding will then be fed into the first LSTM, transforming it into a **512d** vector. This will be further processed by a second LSTM, outputting hidden representations with a dimensionality of **512** each.

Fully-Connected Layer. In order to generate the desired output vectors of size **768**, each hidden representation will be processed through two fully-connected (linear) layers. The first linear layer transforms the input into a **512d** vector, which will then be further transformed by the second layer into the **768d** output vector. *Hyperbolic tangent (tanh)* is used as activation function.

Custom Embeddings. For the purpose of generating custom embeddings for each document, the hidden representation at the last sequence step will be extracted (see Figure 5.5 for reference).

5.3.4 Training Process

For the end-to-end training of the network, *Mean Squared Error* as defined in Equation 4.2 will be used as loss function. The data set is further split into training and validation set, following the common practice for Machine Learning experiments.

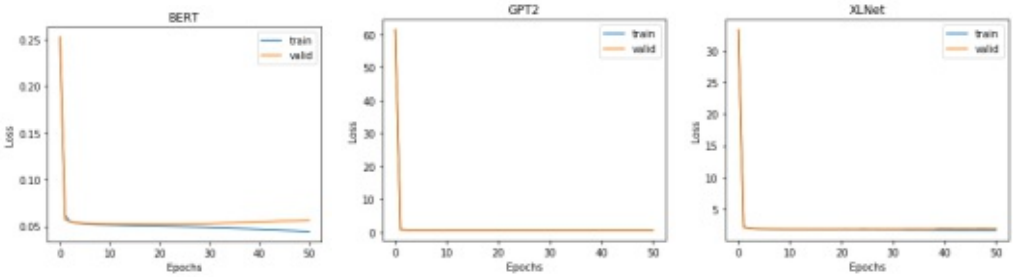

Figure 5.6: Loss Curves for different Embedding Modules

In Figure 5.6, the loss per epoch for the training and validation data set is presented for the different embedding modules. All three models show similar behaviour when it comes to convergence, the only difference is that GPT2 and XLNet start with a higher validation loss before training. Furthermore, we can clearly see the models overfit onto the training data - especially in the case of BERT - which can be seen by the steadily decreasing loss for training data, whereas the validation error starts rising again after a few epochs.

When being compared to each other in a single plot (see Figure 5.7), the difference in the validation error before training as well as the initial error on the training set is presented more clearly. XLNet performs the worst in terms of loss, while starting off better than

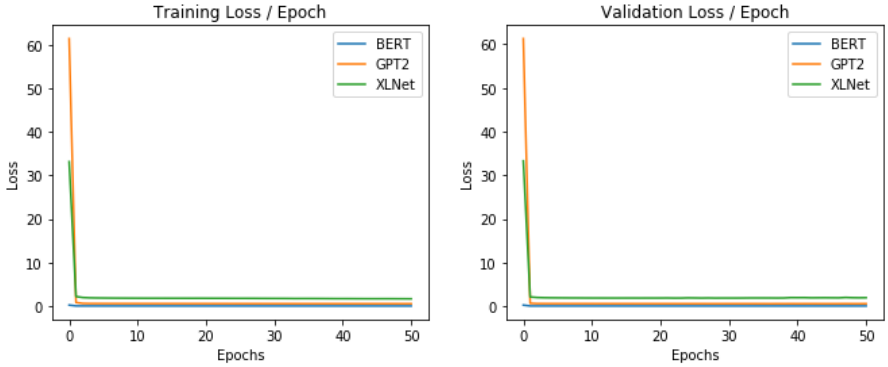

Figure 5.7: Comparison of Train and Validation Error

the GPT2 based model. The model which uses BERT as embedding module achieves the best results for both training and validation error.

5.3.5 Evaluation

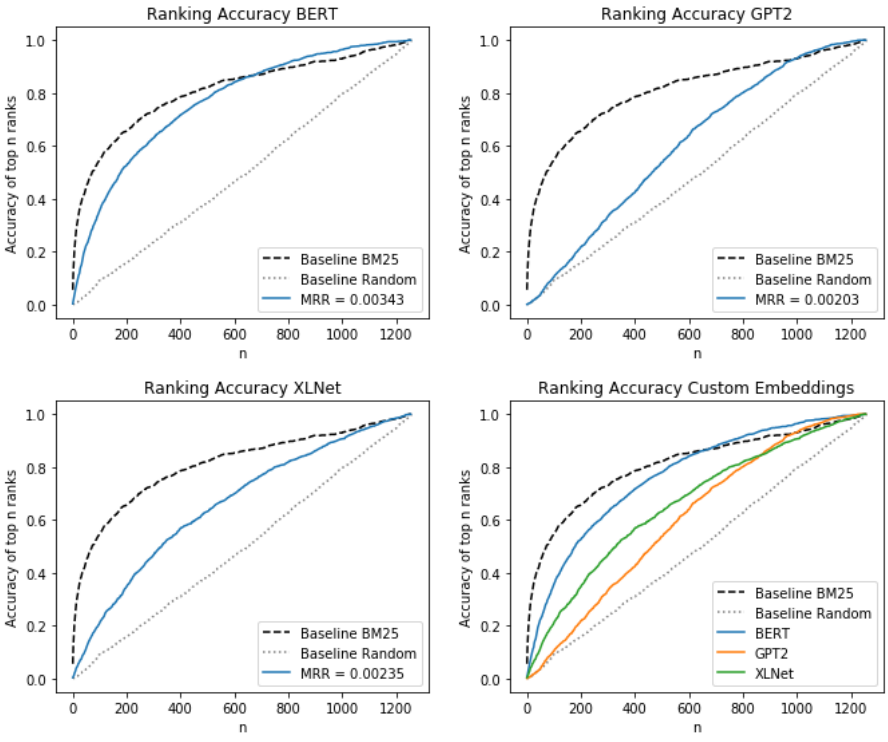

Figure 5.8: Performance of different Embedding Modules

For the evaluation of the model, the test set will be fed into the model. However, as explained in the architecture section, we disregard the predicted output sequence and

take the last hidden representation as custom embedding for the respective document. The resulting feature vector for the abstract will then be used to score all feature vectors representing the research profiles using the *Angular Similarity* function.

As depicted in Figure 5.8, BERT outperforms GPT2 and XLNet, showing a significant improvement on our random scoring baseline. All three of the presented models fail to achieve the performance of the BM25 baseline and are only able to catch up in the later ranks of the evaluation. Out of all models. the network based on BERT as embedding module comes the closest to our state-of-the-art baseline.

5.4 Fully-Connected Autoencoder (FC AE)

For the next experiment, we will present a model based on the *Autoencoder* architecture. Autoencoders are neural networks which can be used for a set of different tasks such as denoising data or generating unseen data, and can be found in many different variants of implementations [DLLK18]. The basic underlying idea of an Autoencoder, however, is dimensionality reduction of a given input vector while preserving as much information as possible to reconstruct the original vector.

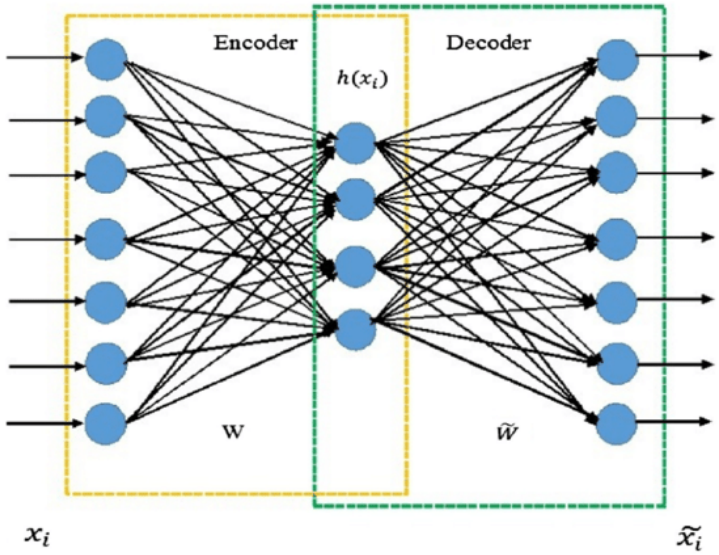

Figure 5.9: Simple Architecture of an Autoencoder [AWN18]

In Figure 5.9, we can see a very simple architecture of an Autoencoder, where x_i represents the input vector, $h(x_i)$ the transformed input with reduced dimensionality, and $\tilde{x_i}$ the reconstructed output vector. The part of the architecture transforming x_i into $h(x_i)$ is referred to as *Encoder*, whereas the reconstruction part from $h(x_i)$ to $\tilde{x_i}$ is called

Decoder. We will use a model based on the Autoencoder architecture to generate custom embeddings of the documents.

5.4.1 Learning Objective

As described in the previous section, an Autoencoder architecture is used to reduce the dimensionality of any given input vector x_i while maintaining its semantic information. The learning objective for our model will be the same as for any other Autoencoder: First, it takes the document embedding in the form of a feature vector x_i as input and processes it through the *Encoder* part to generate the hidden representation $h(x_i)$. Afterwards, it will transform $h(x_i)$ into the output vector $\tilde{x_i}$ using the *Decoder* architecture. By calculating the reproduction error between $\tilde{x_i}$ and x_i, the network will be able to learn and adjust its weights accordingly.

5.4.2 Data Set

The Autoencoder architecture uses the same feature vector it takes as an input also for the respective target. For that reason, the base data set has to be adapted to meet the presented criteria. The change in the data set is a very simple one as all we need is a single document per data point as this will serve both as the input and the target. Thus, the resulting data set is represented by the list of all available documents.

5.4.3 Architecture

The chosen model architecture for this experiment uses three different components: (1) the embedding module including the aggregation module for generating a single representation for a given document (referred to as document embedding), (2) the Encoder as well as (3) the Decoder.

Embedding Module. Basis for the embedding module is again either BERT, GPT2 or XLNet, which takes a given document as input and transforms it into a sequence of **768d** sentence embeddings (referred to as *Token* in Figure 5.10). In order to create a feature vector for the whole document, the aggregation module averages across each dimension of all sentence embeddings and outputs a single **768d** embedding vector.

Encoder. The Encoder of our model is built using two fully-connected layers. The first layer takes the **768d** document embedding as input and transforms it into a vector with a reduced size of **448**. The second layer then processes the vector into a **128d** feature vector, which represents the hidden representation $h(x_i)$. The activation function in-between the layers is *tanh*.

Decoder. Following the Encoder architecture, the Decoder aims at reproducing the document embedding by mirroring the preceding architecture. Therefore, the **128d**

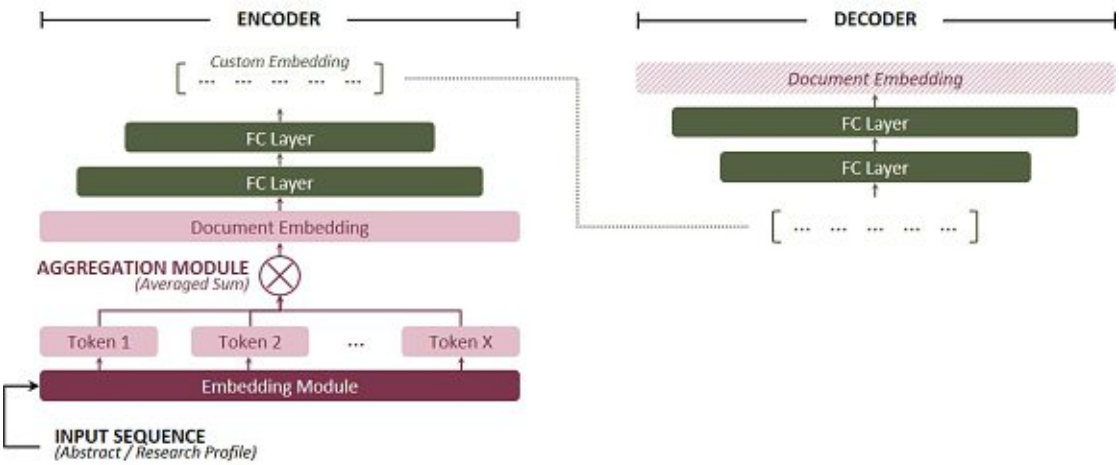

Figure 5.10: Architecture of Fully-Connected Autoencoder

hidden representation will be fed through the first fully-connected layer to generate a **448d** vector. Afterwards, the second fully-connected layer transforms its input back into a **768d** output vector. Again, *tanh* is used as activation functions in-between the layers.

Custom Embeddings. As we aim to generate comparable embeddings based on the documents, we will use the hidden representation $h(x_i)$ - generated by the Encoder module - as the respective custom embedding.

5.4.4 Training Process

To calculate the reproduction error of input and output vector, we will use *Mean Squared Error* as loss function for the training of the network. The data set, again, is split into training and validation set to monitor performance throughout the process.

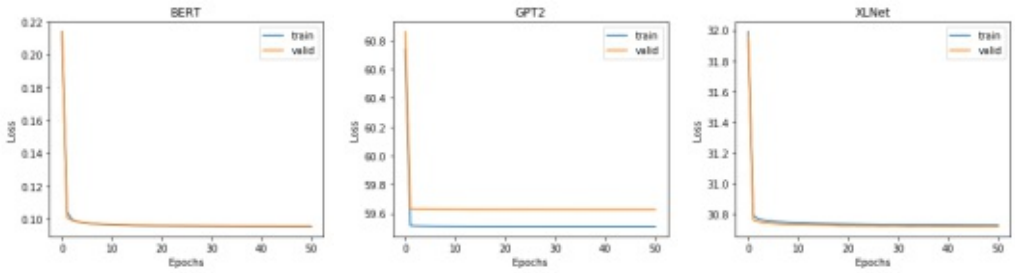

Figure 5.11: Loss Curves for different Embedding Modules

In Figure 5.11, we can observe that all three models converge rapidly for both, training and validation set. While the model based on BERT embeddings starts off with a very

small error and is able to further improve on that, the models implementing GPT2 and XLNet seem to struggle with the given learning objective.

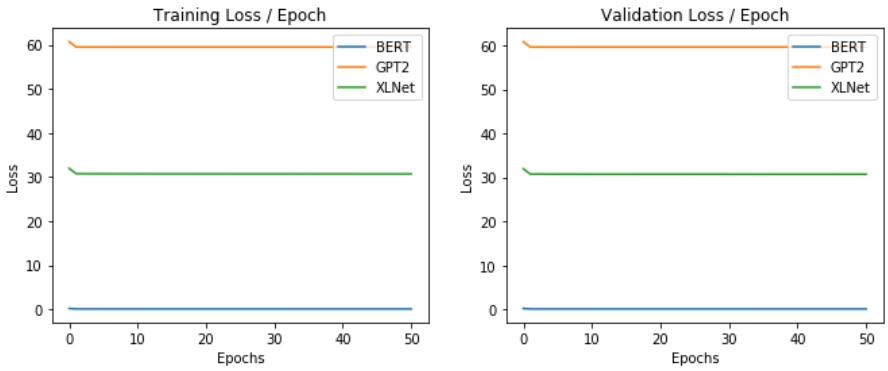

Figure 5.12: Comparison of Train and Validation Error

Comparing the different models side-by-side for the training as well as the validation data set as seen in Figure 5.12, the difference in the calculated loss per epoch clearly shows that BERT is the best performing model. The fast convergence with little learning effect on the models might imply that the chosen architecture does not transfer well to the defined problem. Another reason for the poor performance could be the rather simple approach of aggregating the sentence embeddings into a single document embedding.

5.4.5 Evaluation

The evaluation of our model will be done using only the Encoder architecture on top of the embedding module. The embedding module will take a given document as input, transform it into a single document embedding and feed it further into the Encoder. This will result in the hidden representation vector, which will be calculated for both, abstracts and research profiles. Afterwards, for each abstract, all research profiles will be measured against it using the *Angular Similarity* function.

The network based on BERT embeddings is the best performing model on our evaluation task, as seen in Figure 5.13. The model achieves a higher MRR, however, performs worse at earlier ranks. With GPT2 and XLNet only performing better at very late ranks, the BM25 baseline is the best model for earlier ranks.

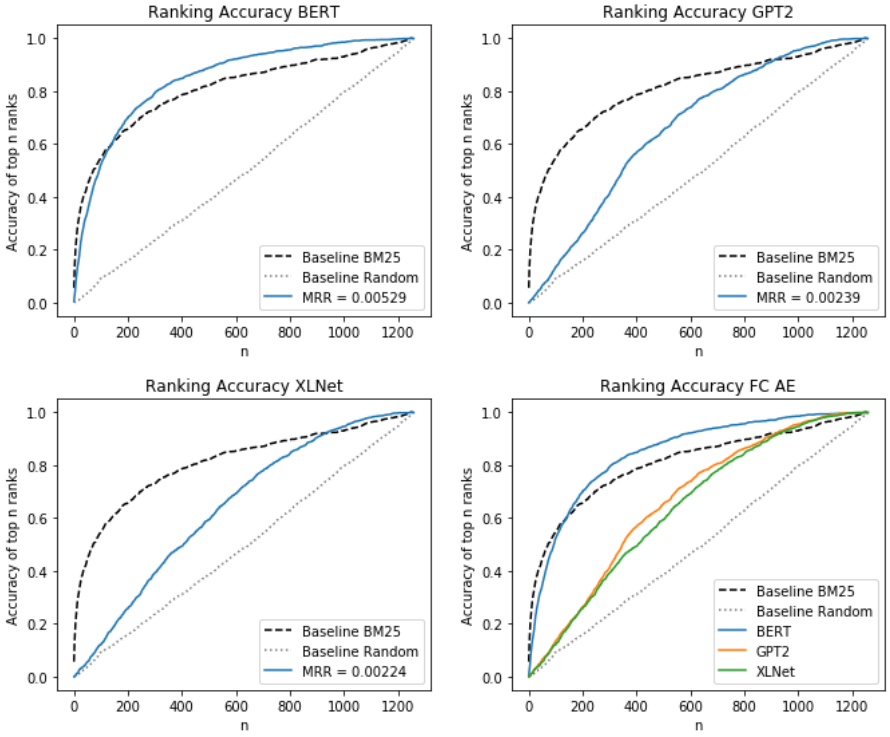

Figure 5.13: Performance of different Embedding Modules

5.5 LSTM-based Autoencoder (LSTM AE)

Following the idea of the first two experiments, we want to combine the principle of dimensionality reduction with sequence processing. Therefore, the model presented in this section will be an *LSTM-based Autoencoder*. The concept is clear: We want to take a sequence of sentences as input and use the Encoder-Decoder architecture to compress them at first, and then reproduce the original sequence again.

Compared to the simple Autoencoder described in the previous subchapter, the model is intended to learn its own aggregation of sentence embeddings within the Encoder architecture, hoping to preserve the semantic text information in a better way and achieve better results.

5.5.1 Learning Objective

Similarly to the learning objective of the simple Autoencoder, we now want to use the model to process a sequence S of different sentence embeddings s_i to s_n to achieve a hidden representation $h(S)$ with lower dimensionality. As the input to the Encoder is not a single feature vector anymore, the Decoder needs to be fed with additional information - the expected sequence length n. To achieve this goal while maintaining a network

architecture which is still differentiable, i.e. it is still able to learn using gradient descent and backpropagation, the hidden representation $h(S)$ is simply fed into the recurrent Decoder n times. The resulting output vector is then a sequence $\tilde{S}$ containing individual embeddings $\tilde{s_i}$ to $\tilde{s_n}$. By calculating the reproduction error at each time step t - that is calculating the error between s_t and $\tilde{s_t}$ - the network will be able to learn.

5.5.2 Data Set

The architecture for the LSTM-based Autoencoder follows the same structure as the Autoencoder, hence, we need to change our base data set to meet the required input criteria. Therefore, we will again take the list of all available documents as data set, as a single data point represents both, the input and the target.

5.5.3 Architecture

The implementation of the LSTM-based Autoencoder is done using the same three components as for the simple Autoencoder: (1) the embedding module which transforms our given input sequence into sentence embeddings, (2) the Encoder architecture, and the (3) Decoder architecture. However, as we are now processing sequences within the Encoder-Decoder part, the building blocks of each component have changed.

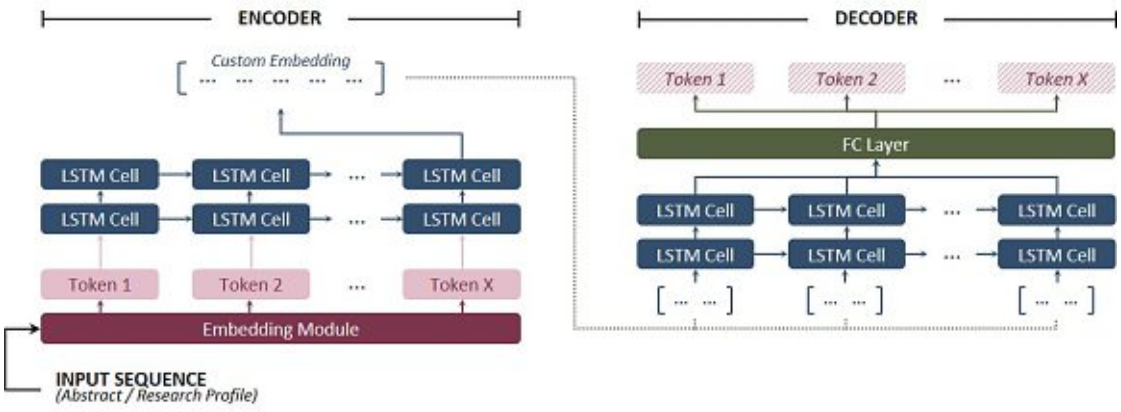

Figure 5.14: Architecture of LSTM-based Autoencoder

Embedding Module. A document will be fed into the first module, the embedding module, which is based on either BERT, GPT2 or XLNet. The embedding module will then transform each sentence of the given document into a **768d** sentence embedding (referred to as *Token* in Figure 5.14). The resulting embedding vector will then be further processed by the Encoder.

Encoder. The Encoder consists of two stacked LSTMs, taking a sentence embedding at each sequence step as input. The first LSTM takes the **768d** embedding vector and

transforms it into a **448d** vector. Subsequently, the second LSTM processes the given vector to generate a hidden representation with reduced size of **128**.

Decoder. Following the Encoder, our Decoder has the purpose of reproducing the original sequence as close as possible. Therefore, as mentioned already in the beginning of this subchapter, the generated **128d** hidden representation will be fed n times into the Decoder architecture (n being the length of the original sequence). In order to process a sequence of inputs accordingly, the Decoder is implemented using two stacked LSTMs, followed by a single fully-connected layer for the output. The first LSTM takes each **128d** hidden representation and generates another **128d** vector at each time step. This vector will then be transformed by the second LSTM, resulting in a **448d** feature vector. The full batch of n different feature vectors is then forwarded into the linear layer. The output matrix of this operation is of size $\mathbf{n} \times \mathbf{768}$, where the first dimension represents the original sequence length n and the second dimension the reproduced sentence embedding $\tilde{s}_i$.

Custom Embeddings. In order to evaluate the performance of this model, we will generate a custom embedding for each document - abstract and research profile - using the hidden representation $h(S)$ calculated by the Encoder architecture.

5.5.4 Training Process

Similarly to the training process of the simple Autoencoder, we will use *Mean Squared Error* as loss function for the training of the LSTM-based Autoencoder. The loss function is used to calculate the reproduction error between the input and output sequence of the model. Following previous guidelines, the data set is split into training and validation set.

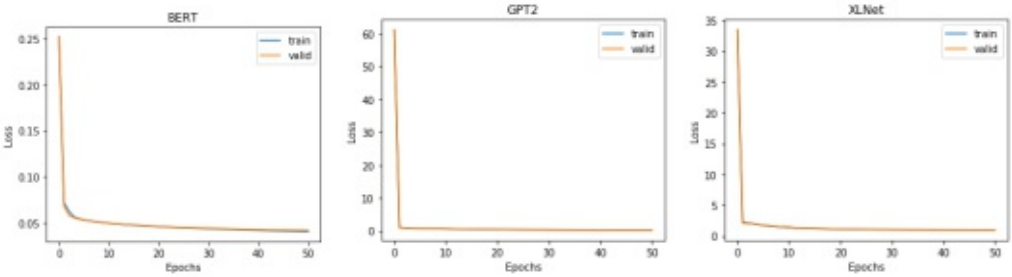

Figure 5.15: Loss Curves for different Embedding Modules

As with the fully-connected Autoencoder, all three different models of the LSTM-based Autoencoder - BERT, GPT2 and XLNet based - converge very quickly with respect to the calculated loss per epoch (see Figure 5.15). However, with this experiment, GPT2 and XLNet both achieve lower error scores which are closer to the one of the BERT

model. In all three cases, the difference between validation and training data set is very small, and no extreme overfitting can be observed.

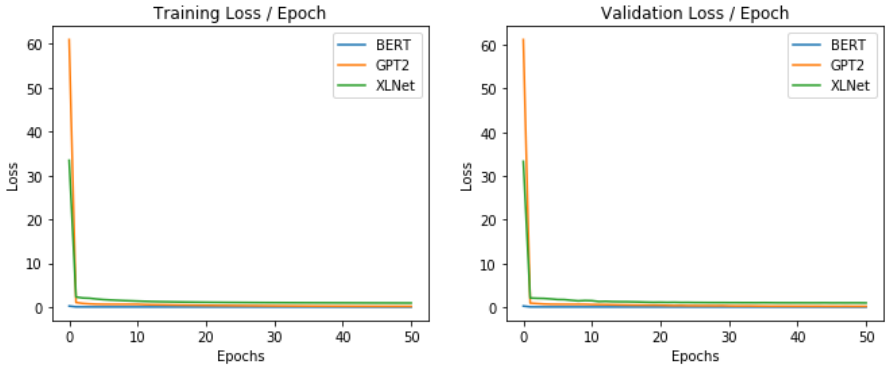

Figure 5.16: Comparison of Train and Validation Error

In Figure 5.16, we can further see that while the three different models start with significant differences in the loss at epoch 0, they meet after a few epochs. While BERT is still the best performing model, the LSTM-based Autoencoder achieved better overall results in the training process than the fully-connected Autoencoder. This might imply that the use of the aggregation module in the fully-connected architecture indeed hinders the model to preserve semantic information.

5.5.5 Evaluation

For the evaluation of the LSTM-based Autoencoder, the hidden representation of the Encoder will be used for both, abstracts and research profiles. Subsequently, each abstract will be compared to all available research profiles using the *Angular Similarity* function, resulting in a ranked list of potential reviewers.

As described in the previous section, the LSTM-based Autoencoder achieves better results for the main learning objective of compressing and reproducing the input vector compared to the fully-connected Autoencoder. However, when it comes the problem description of this thesis, the evaluation of the model shows rather poor performance. In Figure 5.17, we can see that all of our three models achieve lower accuracy throughout almost all ranks compared to BM25. In the earlier ranks, even our random baseline outperforms the models based on GPT2 and XLNet, which is a clear indicator that the learned architecture does not transfer well to the defined problem of this thesis. The network based on BERT as embedding module still significantly outperforms the other models. The overall performance, although, is not as high as with the fully-connected Autoencoder.

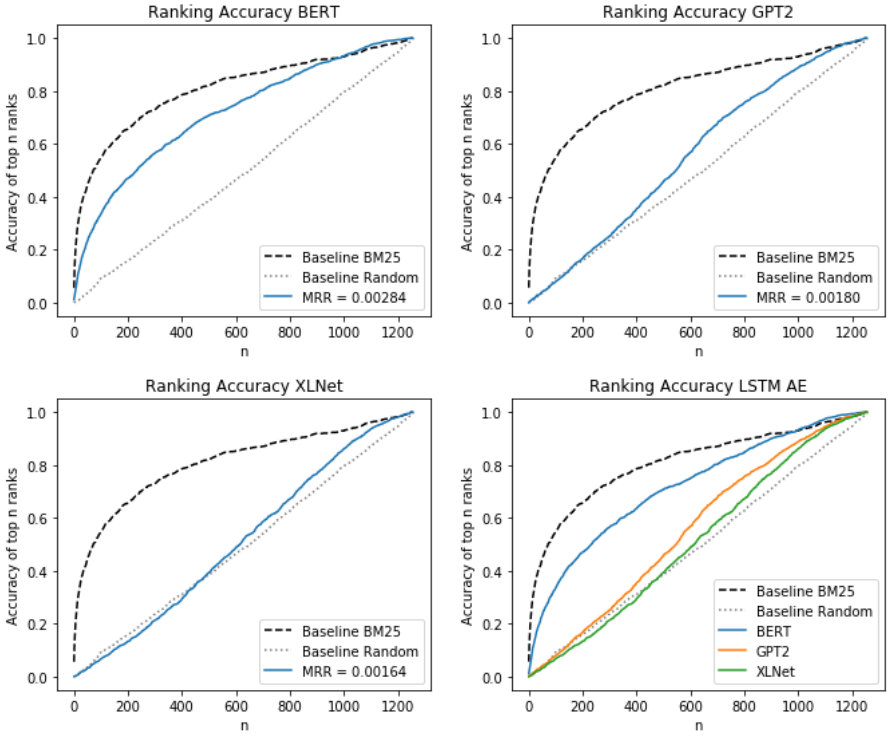

Figure 5.17: Performance of different Embedding Modules

5.6 LSTM-based Dual Encoder

In the previous experiments, we used a single encoding architecture for both types of documents, abstracts and research profiles. Due to the structural differences of those two document types, a single Encoder might not achieve best possible results. Therefore, with the inspiration of the *Siamese Neural Network* architecture presented in [RG19], this experiment aims to improve previous results by using two different encoding modules.

Siamese Neural Networks, as depicted in Figure 5.18, take two different vectors as input and feed them through their respective encoding architecture (referred to as *Network 1* and *Network 2*). The word *siamese* describes that both these networks share the same weights, i.e. it is implemented as one single network. The resulting vectors will then be merged or compared (denoted with S for *Similarity*) and generate a single output vector. In the scope of [Koc15], the output vector represents the similarity between two given input pictures. As our model expects text bodies as input, the output will represent the semantic similarity between two documents.

For the purpose of this experiment and due to the above stated difference in document types, we propose a similar architecture but without shared weights.

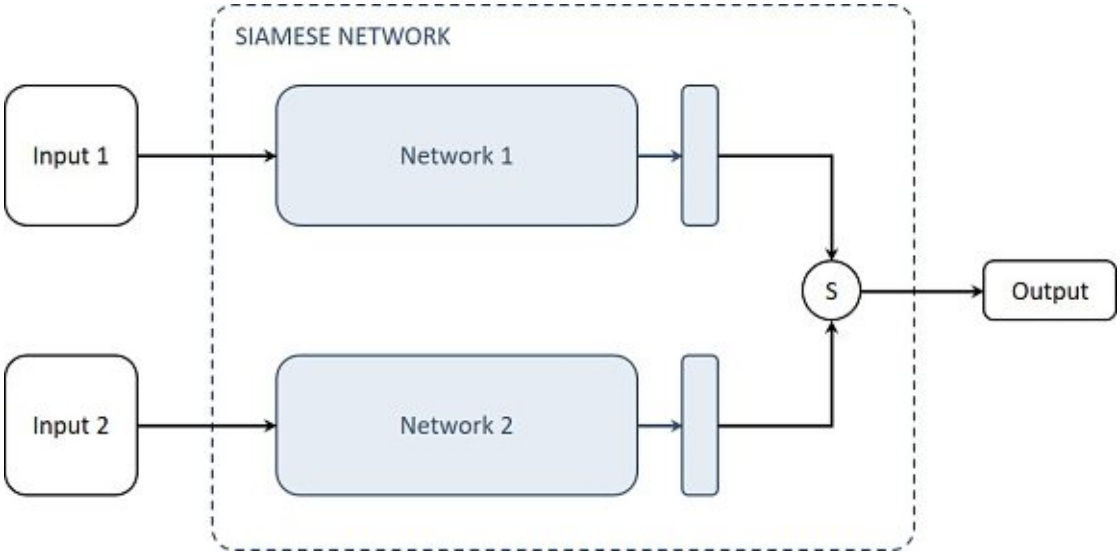

Figure 5.18: Basic Architecture of a Siamese Neural Network

5.6.1 Learning Objective

As described in the previous section, the proposed architecture aims to calculate the similarity between two given input vectors x_1 and x_2. Therefore, we need to define a new learning objective for the overall model. Given both inputs x_1 and x_2, the model will feed them through two different networks respectively to generate the hidden representations $h_1(x_1)$ and $h_2(x_2)$. Both vectors will then be compared with each other using a function to measure vector similarity, returning a **1d** output vector which represents the similarity score in the range $[0, 1]$. A score of 1.0 is achieved when $h_1(x_1) = h_2(x_2)$, while a score of 0.0 represents dissimilarity. If trained correctly, the model will then act as custom similarity function to score each research profile in respect to a given abstract.

5.6.2 Data Set

The proposed architecture expects two documents as input in order to calculate the similarity. The presented base data set includes abstracts, research profiles, and the respective connection in form of the target. However, a new target is also required, as the output of the network is the similarity score in the range $[0, 1]$.

Therefore, each abstract will be paired with the research profile defined by its target, representing a pair of similar documents with a target of 1.0. Without having data points as samples for dissimilarity, the network will most likely learn to maximize the output for any given input pair. Thus, we will make use of the method for negative sampling as described in Chapter 3.5.2 to populate the data set with further document pairs with a target of 0.0. To summarize, for each abstract we will include two samples x_{pos} and x_{neg}

in the new data set:

$$x_{pos} = (abstract, d_{pos}, 1.0) \tag{5.1}$$
$$x_{neg} = (abstract, d_{neg}, 0.0) \tag{5.2}$$

where d_{pos} is the research profile connected to the abstract by its original target, and d_{neg} is the research profile retrieved by our method for negative sampling.

5.6.3 Architecture

The architecture for the LSTM-based Dual Encoder is built using several individual components: The main component of the architecture is the mirrored neural network structure consisting of (1) the embedding module to retrieve sentence embeddings for both, abstract and research profile, and (2) a recurrent layer represented by two stacked LSTMs. Afterwards, the hidden representations of both networks will be processed by (3) the similarity module to generate the final output vector.

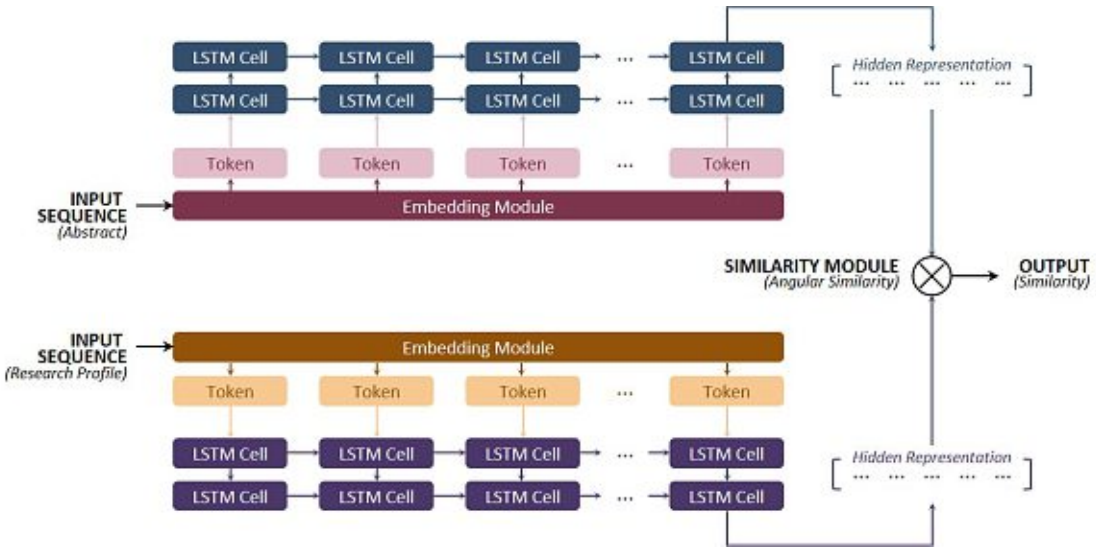

Figure 5.19: Architecture of LSTM-based Dual Encoder

Embedding Module. The first building block of the mirrored neural network architecture is the embedding module. As with previous experiments, this is represented by either BERT, GPT2 or XLNet. Both networks will be based on the same embedding module, i.e. in the case of BERT, it will be used to extract sentence embeddings for the input of both networks, i.e. abstract and research profile. The first network will take the abstract as input, the second one any given research profile. Each embedding module

will output a **768d** sentence embedding at each time step.

Recurrent Layer. Each individual sentence embedding will then be forwarded through the recurrent layer of the respective network, consisting of two stacked LSTMs. The first LSTM takes the **768d** feature vector as input and compresses it into a **448d** vector. The output of the second LSTM will be a **448d** vector as well, however, only the hidden representation at the last sequence step will be used. Each LSTM cell at time step t will pass its activation as input to the LSTM cell at time step $t+1$, following the design of a recurrent neural network. At the end of the sequence, the hidden representation is being calculated based on the aggregation of all previous activations. Thus, the recurrent layer in our model represents a many-to-one LSTM architecture (see Figure 5.4 for reference), where a given input sequence will be transformed into a single output vector.

Similarity Module. After generating both hidden representations for the abstract and the research profile, the similarity module will compare these two vectors, calculating the respective similarity score. For the LSTM-based Dual Encoder we will use the same similarity module as for the evaluation process, the *Angular Similarity* function.

5.6.4 Training Process

The loss function for the training process differs to the one we used for previous experiments. All of the previous models aim at generating custom embeddings, where the target vector(s) is based on the input vector(s). However, the LSTM-based Dual Encoder calculates a single similarity score of two given input documents, i.e. the purpose of the network is to classify two documents as either similar or dissimilar. Thus, instead of using *Mean Squared Error*, we will use *Binary Cross Entropy* as loss function.

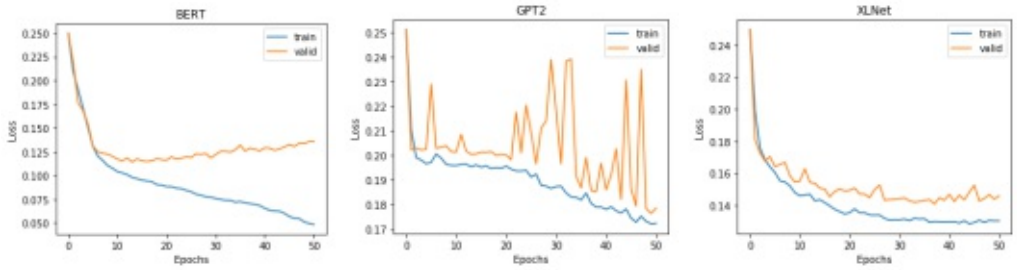

Figure 5.20: Loss Curves for different Embedding Modules

Interestingly, all three different models (BERT, GPT2 and XLNet) behave differently when it comes to the training of the neural network (as seen in Figure 5.20. The model based on BERT as embedding module shows a loss curve very typical for overfitting. While both, training and validation error, continue to decrease over the first epochs, the validation error starts to increase again after approximately 15 epochs. In contrast to BERT, the models using GPT2 and XLNet do not seem to overfit on the provided training

set. the XLNet model slowly converges over the duration of 50 epochs, whereas GPT2 starts to heavily fluctuate in the calculation of the validation error after approximately 20 epochs. Even though the validation error seems to be unstable, the model performance seems to keep improving without converging after 50 epochs.

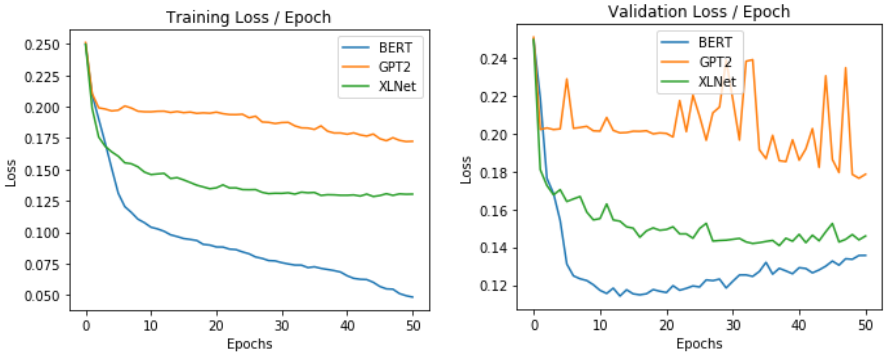

Figure 5.21: Comparison of Train and Validation Error

However, when comparing to the overall picture as presented in Figure 5.21, GPT2 ranks last when it comes to the prediction of unseen data. At the end of 50 epochs, XLNet and BERT seem to achieve similar loss values, over the course of all epochs, though, BERT clearly is the best performing model with a validation error of 0.11419 at epoch 13.

5.6.5 Evaluation

In contrast to previous experiments, the LSTM-based Dual Encoder itself functions already as a similarity scoring module. Hence, there is no need for a separate similarity function in order to evaluate the model on our test data set. Each abstract will be paired with all research profiles and used as input for the model, which then calculates a similarity score for the respective research profile. After scoring all profiles for the respective abstract, a ranking list with descending similarity scores will be computed and used for the evaluation plots.

As seen in Figure 5.22, the LSTM-based Dual Encoder is the first model which achieves an increased performance compared to BM25 for all three different embedding modules (based on the respective MRR). While GPT2 and XLNet are closer to the MRR of our BM25 baseline, the BERT based model outperforms the baseline with an MRR increase of a factor greater than 2. However, BM25 still achieves better performance than all three models at early ranks.

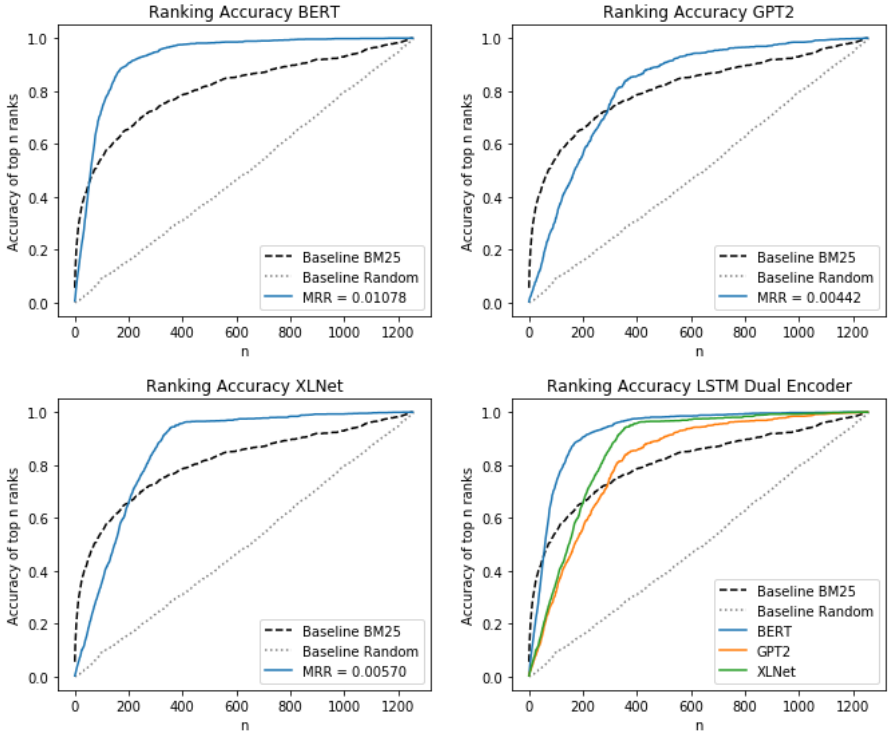

Figure 5.22: Performance of different Embedding Modules

5.7 Siamese Angular LSTM

Due to the good overall performance of the LSTM-based Dual Encoder, as introduced in Chapter 5.6, this experiment is designed as comparison between the initial idea of a *Siamese Neural Network* using shared weights and the implemented adaptation using two independent networks. For further details on the basic architecture of *Siamese Neural Networks*, please refer to Chapter 5.6.

The main purpose of this model is - similar to the LSTM-based Dual Encoder - to act as a similarity scoring function between two different documents.

5.7.1 Learning Objective

The Siamese Angular LSTM expects two input documents, in our case an abstract and a research profile. As defined in Section 5.6.1, the siamese network architecture will generate two hidden representations $h(x_1)$ and $h(x_2)$ based on both input vectors x_1 and x_2. We want to point out that in the case of this model, the hidden representation is denoted using h only instead of h_1 and h_2 because both neural networks will behave in the same manner due to the shared weights. Afterwards, the model will calculate a similarity score between the two given input documents.

5.7.2 Data Set

The data set will be adjusted similarly to the data set defined in Section 5.6.2. To summarize, each available abstract will be paired with the respective research profile of its original target, and assigned with a new target similarity of 1.0. Additionally, using the presented negative sampling method, the same abstract will be paired with the newly retrieved research profile and assigned with the target 0.0 to represent dissimilarity.

5.7.3 Architecture

The Siamese Angular LSTM is based on the architecture of a *Siamese Neural Network*. Thus, the model consists of three main building blocks: The siamese networks - implemented by defining only one neural network which is then used for both inputs in parallel - consists of (1) the embedding module to generate sentence embeddings for the given input document and (2) the recurrent layer built with two stackes LSTMs. The generated hidden representations of this network will then be compared and scored by (3) the similarity module.

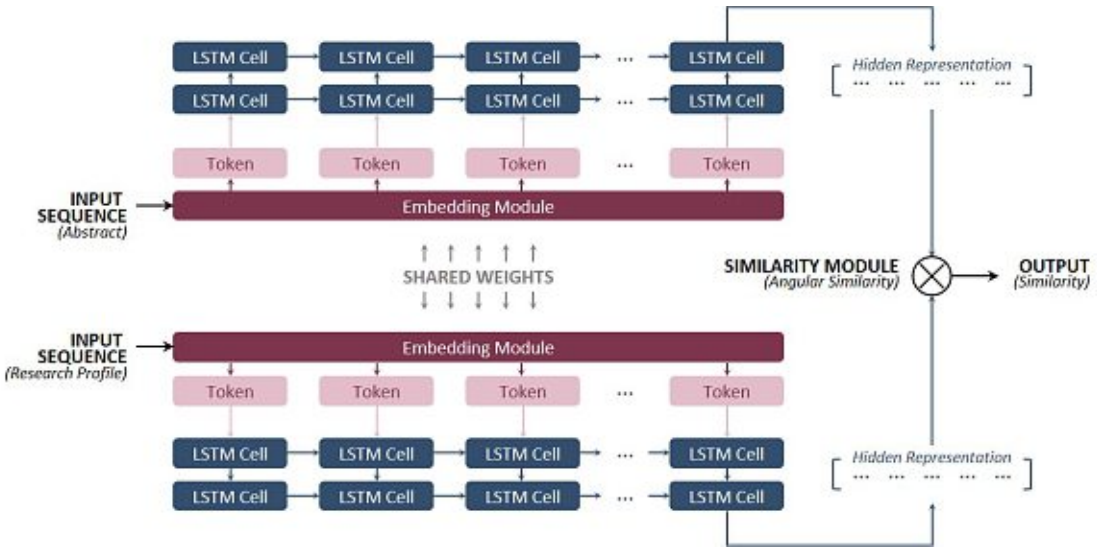

Figure 5.23: Architecture of Siamese Angular LSTM

The technical details of each individual component are the same as defined in Section 5.6.3, where we introduced the architecture of the LSTM-based Dual Encoder. In short, the **embedding module** generates a sequence of **768d** sentence embeddings for each of the two documents. Each sentence embedding is then further processed by the **recurrent layer**, where the first LSTM transforms the embedding into a **448d** vector. The **448d** hidden representation at the last sequence step of the second LSTM is then being used as input for the similarity module. The **similarity module** is represented by the *Angular Similarity* function and outputs a single **1d** output vector in the range $[0, 1]$.

5.7.4 Training Process

For the training process of the Siamese Angular LSTM, we will use *Binary Cross Entropy* as loss function. Reasoning for using *Binary Cross Entropy* over *Mean Squared Error* is the same as defined for the LSTM-based Dual Encoder: We want to classify two given input documents as either similar, with a target of 1.0, or dissimilar, with a target of 0.0.

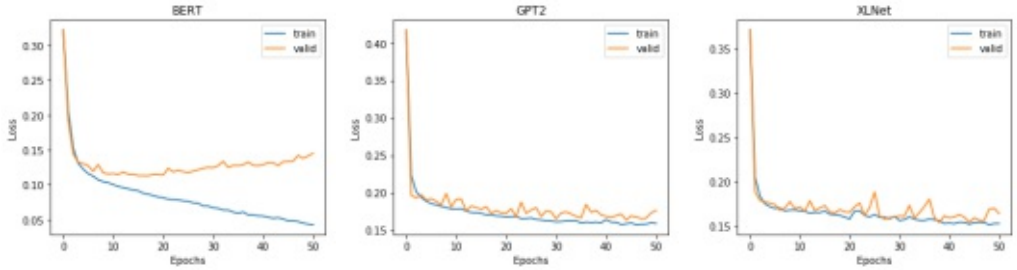

Figure 5.24: Loss Curves for different Embedding Modules

In Figure 5.24, we can observe the performance on the training process using our training and validation data set. The models using GPT2 and XLNet as embedding modules behave in a very similar manner, slowly converging over the duration of 50 epochs without any unexpected fluctuations in the calculated loss per epoch. Our BERT based model learns faster as seen in the decrease of the loss curve of the training set. However, after about 10 epochs, the model starts to overfit which can be observed by the validation error increasing again.

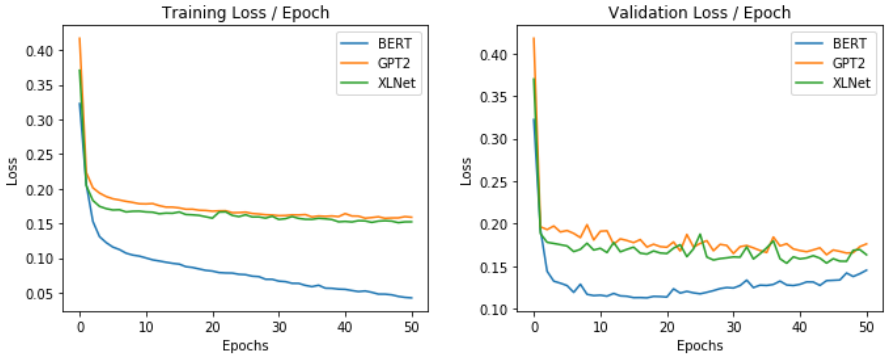

Figure 5.25: Comparison of Train and Validation Error

When comparing all three models to each other (as seen in Figure 5.25, the close performance of GPT2 and XLNet can be observed even more clearly, while XLNet has a slight edge over GPT2. Overall, BERT performs the best out of all three models.

5.7.5 Evaluation

To evaluate the presented architecture on the main problem description of this thesis, we will use the network for both embedding and similarity scoring. Each abstract will be paired with all research profiles as input for the network, which then calculates a similarity score in the range $[0, 1]$ for the two documents. Once all available research profiles are scored for the respective abstract, the list of research profiles will be ranked descending based on their score and used for the evaluation plots.

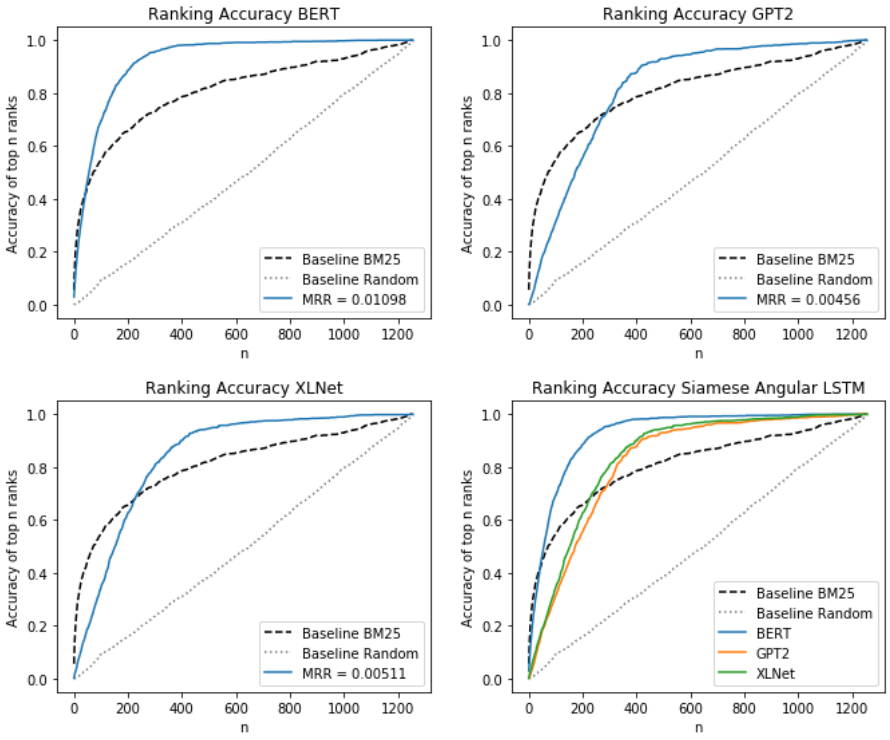

Figure 5.26: Performance of different Embedding Modules

Figure 5.26 shows the performance of all three different models and its comparison. Here we can see that all models manage to outperform our baselines, with BERT being the best performing model as already anticipated within the training process. Compared to the LSTM-based Dual Encoder, the MRR of the BERT based model further increased from 0.01078 to 0.01098. However, the BERT model still does not achieve the performance of the BM25 baseline when it comes the earlier ranks (top 100).

5.8 Siamese Manhattan LSTM

For the final experiment, we will present another architecture based on the principle of *Siamese Neural Networks*. Our previous models show significantly increased performance

compared to the models which are not based on a siamese architecture. As we want to keep the recurrent property of our network due to the input format and the pre-built embedding modules, we are changing the last building block: the similarity module.

So far, we have used *Angular Similarity* as evaluation metric for earlier models and similarity module for our models based on *Siamese Neural Networks*. In [Thy15], the *MaLSTM* is being presented, using a similarity measure derived from the *Manhattan Distance* function. We will call the derived similarity measure *Manhattan Similarity* as defined in Equation 2.6.

5.8.1 Learning Objective

The main learning objective for the Siamese Manhattan LSTM is defined similarly to the Siamese Angular LSTM and the LSTM-based Dual Encoder (for reference see Chapter 5.6.1). The network will take two documents x_1 and x_2 as input, an abstract and a research profile, and will generate two hidden representations $h(x_1)$ and $h(x_2)$. These hidden representations will then be compared and a similarity score in the range of $[0, 1]$ will be calculated as output.

5.8.2 Data Set

As for the learning objective, we will refer to Chapter 5.6.2, which defines the data set adjustment for the LSTM-based Dual Encoder. Each data point is represented as tuple, containing an abstract, a research profile and a target of either 1.0 or 0.0. The respective research profiles will be retrieved by the original target of the abstract for positive samples, and using the negative sampling method for negative samples.

5.8.3 Architecture

Following the previous experiments based on *Siamese Neural Networks*, the Siamese Manhattan LSTM will be built with the same three building blocks: both, (1) the embedding module to extract sentence embeddings and (2) the recurrent layer consisting of two stacked LSTMs, as part of the siamese network, and (3) the similarity module.

In Figure 5.27, we can see that the network is very similar to the architecture of the Siamese Angular LSTM (see Figure 5.23 for reference). The only difference is that we use *Manhattan Similarity* as our similarity function, which is based on the *Manhattan Distance*. The **embedding module** will output a **768d** embedding for each sentence of a respective document, which will then be further transformed into a **448d** vector by the first LSTM and, subsequently, into a **448d** hidden representation by the second LSTM. Afterwards, the **similarity module** takes the hidden representations of both input documents and calculates a **1d** output vector representing the similarity in the range $[0, 1]$.

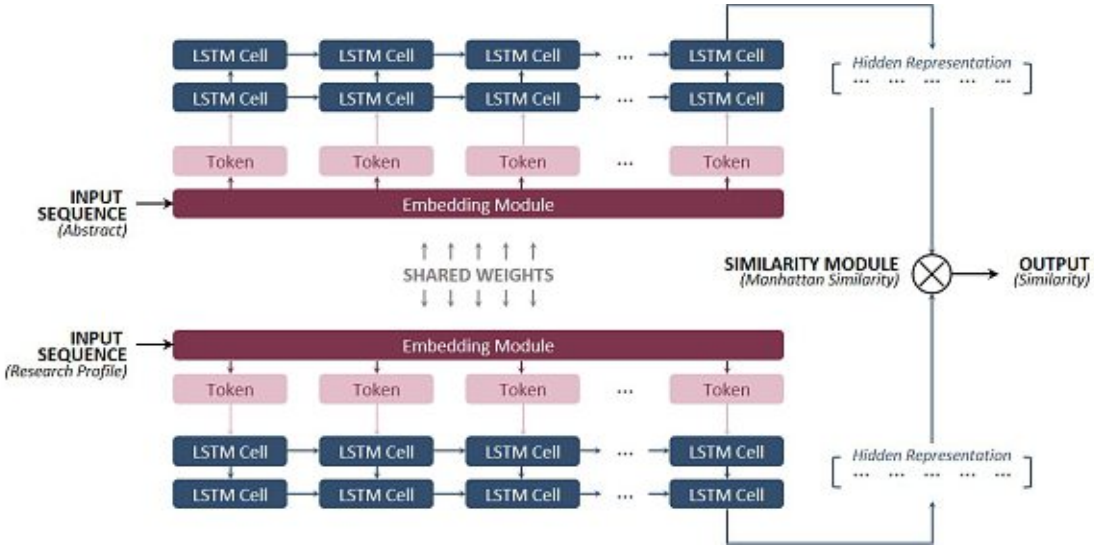

Figure 5.27: Architecture of Siamese Manhattan LSTM

5.8.4 Training Process

For the training of the presented network, we will use *Binary Cross Entropy* as loss function due to the purpose of the model being to classify two documents as either similar, with a target of 1.0, or dissimilar, with a target of 0.0.

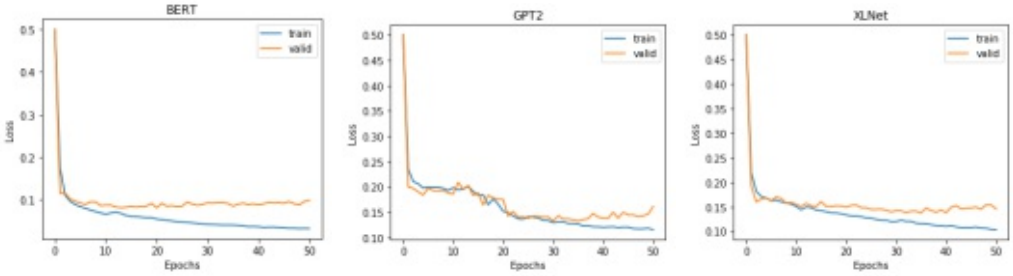

Figure 5.28: Loss Curves for different Embedding Modules

In Figure 5.28, we can observe that the change from *Angular Similarity* to *Manhattan Similarity* as similarity function reduced the amount of overfitting for the BERT based model. Overall, all three models continuously improve their performance on the provided training set over the duration of 50 epochs, whereas the validation error converges rather quickly. The model using GPT2 as embedding module shows interesting behaviour as the loss for both, training and validation set, suddenly starts decreasing again after approximately 10 epochs.

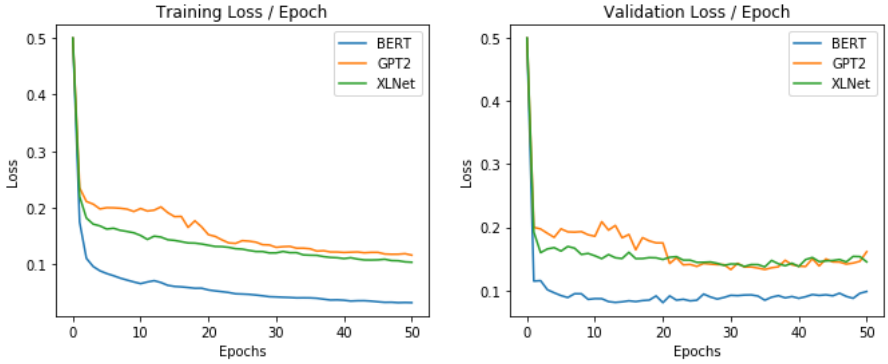

Figure 5.29: Comparison of Train and Validation Error

In the comparison of training and validation error across all three models (see Figure 5.29), we can observe that the BERT based model once again presents itself as best performing model. In contrast to previous experiments, though, GPT2 manages to perform slightly better than XLNet on the prediction of the unseen validation data set.

5.8.5 Evaluation

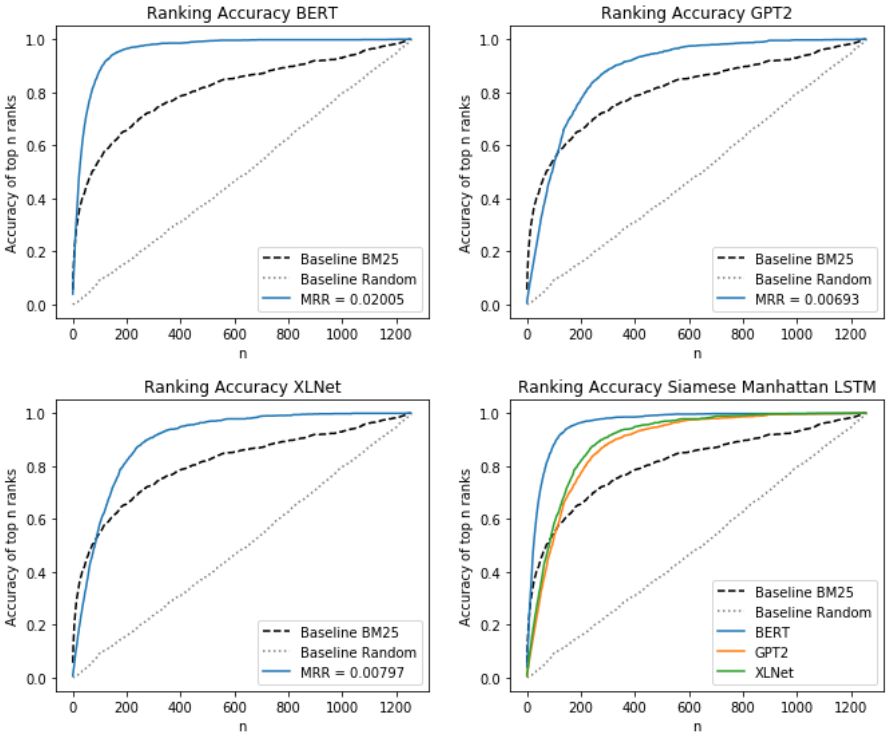

Figure 5.30: Performance of different Embedding Modules

For the evaluation of the Siamese Manhattan LSTM, we use the same approach as defined in Section 5.7.5, i.e. for each abstract, the similarity scores for all research profiles will be calculated. The resulting list of scored research profiles will be ranked in descending order based on the similarity scores.

In Figure 5.30, we see the performance of all three models individually and in comparison to each other. Compared to the Siamese Angular LSTM, all models of the Siamese Manhattan LSTM show an improved MRR, meaning that they outperform our baselines as well. The Siamese Manhattan LSTM using BERT as embedding module achieves the best performance with a MRR of 0.02005, which is almost double the score of the Siamese Angular LSTM (MRR of 0.01098).

Due to the very similar performance of the BERT based model and the BM25 baseline in the very early ranks, we will compare the top ranks based on each individual accuracy score. In Table 5.2, we can see that BM25 starts with a higher accuracy for the first 9 ranks. However, starting with an n of 10, the Siamese Manhattan LSTM achieves a better score (marked in blue), which then rapidly improves over the baseline (as seen in Figure 5.30).

n	BM25	BERT	n	BM25	BERT
1	0.05685	0.04043	**11**	0.25584	0.27037
2	0.10044	0.06443	**12**	0.26784	0.28932
3	0.13076	0.08781	**13**	0.27921	0.30259
4	0.15413	0.11939	**14**	0.28869	0.31775
5	0.17687	0.14403	**15**	0.29500	0.33101
6	0.19583	0.16866	**16**	0.30259	0.34744
7	0.21225	0.18888	**17**	0.30764	0.36639
8	0.21983	0.20972	**18**	0.31585	0.38534
9	0.23436	0.22994	**19**	0.32280	0.40366
10	0.24699	0.25268	**20**	0.32596	0.42008

Table 5.2: Comparison of Siamese Manhattan LSTM (BERT) and BM25 for the top n Ranks based on Accuracy

5.9 Comparison & Concluding Remarks

For our concluding remarks on the performance of all different networks, we will compare each network architecture with the others. In Figure 5.31, we can observe the performance of all individual experiments, including the very first experiment using the embedding modules without any further processing.

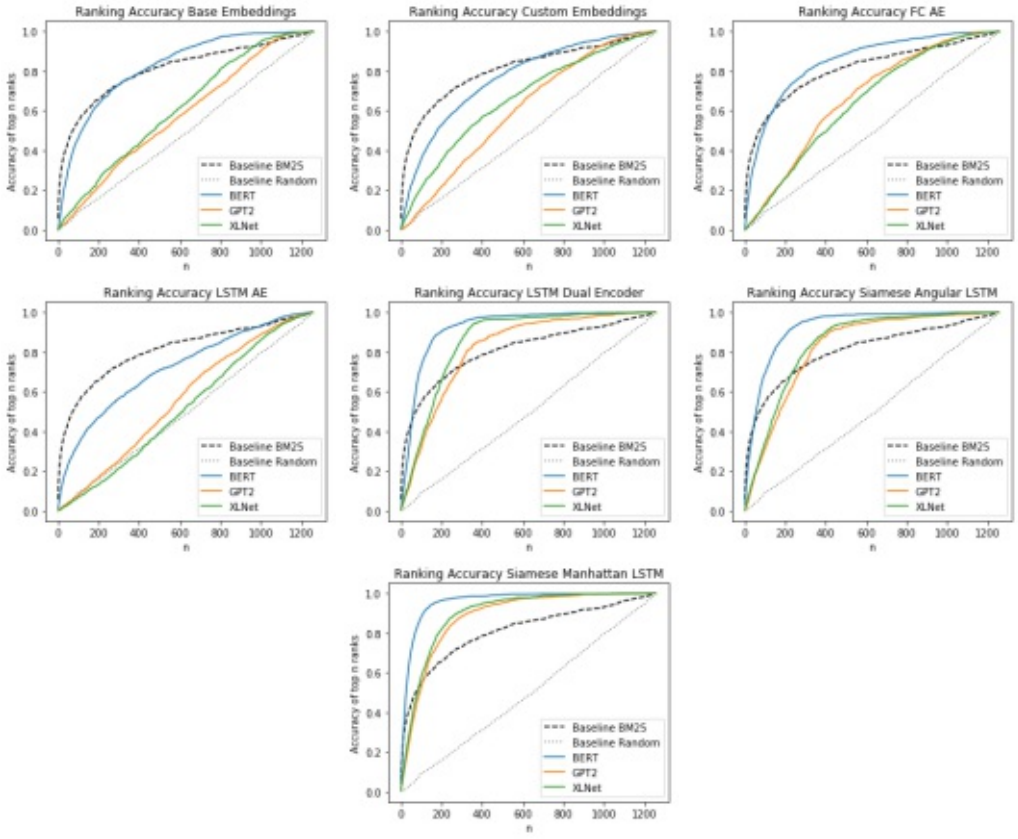

Figure 5.31: Comparison of Model Performance

Overall, the Siamese Manhattan LSTM achieves the best performance across all three different embedding modules, with BERT being the best model. The models based on the *Siamese Neural Network* architecture clearly outperform the first experiments, where we generated custom embeddings for each document. This might be due to the fact that our siamese networks learn to act as a similarity scoring function as their primary learning objective, whereas the other experiments were trained for different purposes.

Table 5.3 presents more details on the performance of individual models by listing their respective MRR. We can observe that our models based on the *Siamese Neural Network* architecture managed to achieve better results than the BM25 baseline, while previous

Model	BERT	GPT2	XLNet
Base Embeddings	0.00458	0.00189	0.00206
Many-to-many LSTM	0.00343	0.00203	0.00235
Fully-Connected Autoencoder	0.00529	0.00239	0.00224
LSTM-based Autoencoder	0.00284	0.00180	0.00164
LSTM-based Dual Encoder	0.01078	0.00442	0.00570
Siamese Angular LSTM	0.01098	0.00456	0.00511
Siamese Manhattan LSTM	**0.02005**	**0.00693**	**0.00797**

Table 5.3: Comparison of Model Performance based on MRR - measured against Random Scoring Baseline (0.00158 MRR) and BM25 Baseline (0.00420 MRR)

experiments achieved a lower score (marked in red). The best MRR score for each embedding module is highlighted in **bold** font, where we can see that the Siamese Manhattan LSTM is the overall best architecture.

However, the evaluation of our experiments was based on the predefined ground truth, i.e. the supervisor for each of abstract. As our final goal is to compute a list of possible reviewers other than the original supervisor, there is an additional need to evaluate the returned ranking list of research profiles. Some of the research profiles used for the training and evaluation process, however, are of poor quality, e.g. very little content on previous research or only visited conferences being listed. Therefore, the performance as depicted in Figure 5.31 and Table 5.3 might be misleading.

CHAPTER 6

Evaluation of Use Cases

In this chapter, we will introduce three different use cases for the previously presented methods. Each use case represents the abstract of a diploma thesis, written by students as part of their graduation in the master's programme *Logic and Computation*.

As motivated in Chapter 5.9, all three use cases will be evaluated manually to measure the respective performance. Manual evaluation means that each trained model will take each abstract as input and compute a ranking of possible reviewers based on the data set defined in Chapter 3. For each ranking list, the top 5/10/20 ranked research profiles will be compared to the abstract in order to determine whether or not it is a relevant match. The results will then be presented in form of a table, listing all models and comparing them using a calculated precision score *Precision @ k* [AGB+19].

Precision @ k (P@k). Let k be the number of documents (research profiles) in scope, starting with rank 1 of the resulting ranking list. The metric *Precision @ k* is defined as

$$P@k = \frac{r}{k} \tag{6.1}$$

with r being the number of relevant documents in respect to the model input (abstract).

Based on the performance observed in Chapter 5, the evaluation will be done using BERT as embedding module for all models. As with former experiments, we use BM25 as baseline model. Additionally, we will use the top 50/100 results from our BM25 model and rerank them using both Siamese Networks. The selection of use cases was limited to topics familiar to the author of this thesis as the manual evaluation process requires basic understanding and background knowledge to mark each document as relevant or

non-relevant. In the following section, each use case will be presented with its title and abstract.

6.1 Use Cases Definition

Use Case 1. Graph-classes of Argumentation Frameworks with Collective Attacks - Properties and Complexity Results [Kö20]

"Abstract Argumentation Frameworks (AFs) constitute a simple formalism to represent and reason over knowledge. The simple way of modelling information just as a directed graph lead to a rapid rise of its popularity. However, the simplicity came with the cost of strict syntactic restrictions, and as it is often inconvenient to formalize certain natural structures in the original notion of frameworks, many extensions have been proposed. We deal with one such extension, namely Argumentation Frameworks with Collective Attacks *(SETAFs), which allow attacks to origin from sets of arguments. In particular, this means that instead of a directed graph, the underlying structure of the framework is a* directed hypergraph.

Many applications motivated the search for efficient reasoning tools for the frameworks, and as most reasoning problems are intractable in general, a more fine-grained complexity analysis was needed. In this thesis we try to identify tractable fragments *for reasoning tasks on SETAFs. In particular, we investigate how certain restrictions on the hypergraph-structure such as acyclicity, symmetry, and bipartiteness can be formulated and whether they allow us to reason more efficiently. We show that the low complexity of acyclicity carries over from AFs to SETAFs, and give the even more general result that this fragment extends to frameworks that have cycles of length* 1*. Moreover we establish tractable fragments in even-cycle-freeness, self-attack-free ϵ-symmetry, and β-bipartiteness, the respective defining properties are introduced in the course of this thesis."*

Use Case 2. A General Framework for Choice Logics [Ber20]

"The topic of preferences is of importance in many areas of research, including computer science, and, more specifically, artificial intelligence. Two formal systems in the literature that are designed for preference handling are Qualitative Choice Logic (QCL) and Conjunctive Choice Logic (CCL). Both of these logics extend classical propositional logic by a non-classical choice connective, with which preferences can be expressed. Instead of evaluating formulas to true or false, formulas in QCL and CCL are ascribed a satisfaction degree, by which interpretations are ranked. In this thesis, QCL and CCL are generalized by the formal introduction of a choice logic framework. Besides showing that QCL and CCL are captured by this framework, several new choice logics, based on new choice connectives, are introduced. Since the specified framework is not very restrictive, and therefore a multitude of different choice logics can be expressed, several classes of choice logics are defined and examined. A notion of strong equivalence between the formulas

of a choice logic is introduced and related to other notions of equivalence. In the course of this analysis, it is proven that for QCL and CCL, our notion of strong equivalence is interchangeable with another notion of equivalence introduced by Brewka et al. in the original QCL paper. Lastly, the computational complexity of different reasoning tasks relevant for choice logics is examined. Although this complexity analysis is conducted in a general manner, it also yields new results regarding QCL and CCL. For example, the main decision problem regarding preferred models is $\Theta 2P$-complete in both of these logics. For the same problem, we show $\Delta 2P$-completeness for Lexicographic Choice Logic (LCL), a logic introduced in this thesis."

Use Case 3. Model Checking Automotive Software Components [Dur20]

"Software systems developed for the automobile industry are currently witnessing a tremendous increase in complexity, happening in concert with an escalation of safety requirements. Ensuring their correctness is a challenging and costly task, which motivates the research for new technological solutions. While not yet broadly adopted in industry, software verification tools targeting C code demonstrate impressive performance improvements every year, which makes them good candidates for ensuring the safety of embedded systems.

This study aims at evaluating the capacity of state-of-the-art verification tools for proving the absence of run-time errors on four software components of various complexity. These real-world components are developed by TTTech, a firm specialized in safety-related automotive solutions.

Firstly, we establish a generic environment model based on the AUTOSAR standard, which is broadly adopted in the automobile industry. This model aims at isolating a component from the rest of the software platform for verification, and uses already-existing specifications defined by the standard. We then check the code using Ultimate Automizer, CPAChecker and CBMC extended with several ideas, such as k-induction or variable range analysis.

Our results show that verification tools are able to successfully prove the absence of errors in three out of four components, and cannot give a definite answer for the most complex one. The most capable verification method is obtained by combining the results of different code analyzes, with CBMC establishing the final verdict using k-induction. For the problematic case, we give insights into the causes of difficulties, and next steps required to overcome them. We conclude that introducing verification tools in the development process can bring positive changes to the general code quality."

6.2 Use Case 1

The first use case, *Graph-classes of Argumentation Frameworks with Collective Attacks - Properties and Complexity Results* [Kö20], will focus on (but not limited to) research profiles which show one or more of the following skills: complexity (theory), argumentation frameworks/theory, non-monotonicity, legal reasoning, hypergraphs, graph properties/theory, knowledge representation, and artificial intelligence in a broader sense.

Model	P@5	P@10	P@20
Baseline BM25	**0.8**	**0.8**	0.5
Base Embeddings	0.4	0.3	0.25
Many-to-many LSTM	0.2	0.1	0.1
Fully-Connected Autoencoder	0.4	0.3	0.25
LSTM-based Autoencoder	0.2	0.1	0.15
LSTM-based Dual Encoder	0.4	0.2	0.15
Siamese Angular LSTM	0.4	0.5	0.4
Siamese Manhattan LSTM	0.0	0.1	0.25
Re-rank Top 50 (Siamese Angular LSTM)	0.6	**0.8**	**0.65**
Re-rank Top 50 (Siamese Manhattan LSTM)	0.6	0.6	0.5
Re-rank Top 100 (Siamese Angular LSTM)	0.6	0.7	**0.65**
Re-rank Top 100 (Siamese Manhattan LSTM)	0.4	0.5	0.6

Table 6.1: Comparison of Model Performance based on P@k

The BM25 baseline model achieves the best results, with **4** relevant profiles in the top 5 and **8** in the top 10. The experiments introduced in Chapter 5, however, achieve only average performance. Especially surprising is the poor performance of the Siamese Manhattan LSTM, which fails to find a good match within the top 5 (marked red in Table 6.1). 6 of our models (highlighed in blue) even present one of the actual supervisors of the original thesis within the top 20 results.

Following the reranking strategy, i.e. combining the top 2 perfoming models as presented in Chapter 5.9 with the BM25 baseline, we manage to outperform our baseline in terms of overall number of relevant profiles and achieve **13** good matches in the top 20. Overall, we can observe that using BM25 as initial ranking model and, subsequently, our Siamese Angular LSTM to rerank the top 50 results, generates the best possible ranking list for the respective user.

6.3 Use Case 2

For our second use case, *A General Framework for Choice Logics* [Ber20], we will use the following (incomplete) list of keywords to manually evaluate the performance of the models: preferences, non-monotonicity, complexity (theory), logic, non-classical logics, knowledge representation, computational properties, and artificial intelligence in a broader sense.

Model	P@5	P@10	P@20
Baseline BM25	**1.0**	**0.9**	**0.65**
Base Embeddings	0.4	0.2	0.25
Many-to-many LSTM	0.2	0.1	0.1
Fully-Connected Autoencoder	0.6	0.4	0.35
LSTM-based Autoencoder	0.2	0.1	0.15
LSTM-based Dual Encoder	0.2	0.2	0.15
Siamese Angular LSTM	0.4	0.5	0.35
Siamese Manhattan LSTM	0.0	0.3	0.35
Re-rank Top 50 (Siamese Angular LSTM)	0.4	0.5	**0.65**
Re-rank Top 50 (Siamese Manhattan LSTM)	0.8	0.6	**0.65**
Re-rank Top 100 (Siamese Angular LSTM)	0.4	0.5	0.45
Re-rank Top 100 (Siamese Manhattan LSTM)	0.6	0.5	0.5

Table 6.2: Comparison of Model Performance based on P@k

In Table 6.2, we can see that BM25 outperforms all other models for the top 10 documents, where **9/10** profiles represent a good match as possible reviewer. As with use case 1, the Siamese Manhattan LSTM does not result in any matches within the top 5 results (highlighted in red), but achieves average results when it comes to overall number of relevant documents in the top 20. The models in blue include the supervisor of the original thesis in the top 20 results, with some of them even presenting one of the actual examiners.

In the bottom rows of Table 6.2, we can observe that our reranking models based on the top 50 results from BM25 achieve top performance for *Precision @ 20*, tied with the BM25 baseline. For this specific use case, however, the plain BM25 model without reranking achieves the best results across all metrics.

6.4 Use Case 3

The final use case presented within this chapter, *Model Checking Automotive Software Components* [Dur20], will focus on research profiles related to computer/software verification and model checking.

Model	P@5	P@10	P@20
Baseline BM25	0.4	**0.6**	**0.5**
Base Embeddings	0.2	0.2	0.2
Many-to-many LSTM	0.0	0.0	0.0
Fully-Connected Autoencoder	0.4	0.3	0.25
LSTM-based Autoencoder	0.2	0.1	0.15
LSTM-based Dual Encoder	0.2	0.1	0.05
Siamese Angular LSTM	0.0	0.0	0.1
Siamese Manhattan LSTM	0.2	0.1	0.1
Re-rank Top 50 (Siamese Angular LSTM)	**0.6**	0.5	0.45
Re-rank Top 50 (Siamese Manhattan LSTM)	**0.6**	0.4	0.35
Re-rank Top 100 (Siamese Angular LSTM)	0.2	0.4	0.35
Re-rank Top 100 (Siamese Manhattan LSTM)	0.4	0.5	0.35

Table 6.3: Comparison of Model Performance based on P@k

In contrast to the other two use cases, the top performing model for *Precision @ 5* is our reranking model based on the top 50 results of BM25. Both models, the Siamese Angular LSTM as well as the Siamese Manhattan LSTM, improve the baseline score to **3** out of 5 good matches. For later ranks, however, BM25 manages to find more relevant documents within the top 10 as well as the top 20. The many-to-many LSTM architecture fails to find any matches (marked in red) and the plain Siamese Angular LSTM only returns **2** relevant documents in the top 20 results.

6.5 Comparison & Concluding Remarks

For the overall comparison of the three presented use cases, we will look at the number of relevant documents retrieved from the top 20 results of each individual model (see Table 6.4). While some models might perform better at earlier ranks than others, the end user might want to look at more than 5 or 10 possible reviewers for his or her thesis. Therefore, we arrive at the assumption that the top 20 research profiles represent the best method for comparability of models.

Model	UC 1	UC 2	UC 3
Baseline BM25	0.5	**0.65**	**0.5**
Base Embeddings	0.25	0.25	0.2
Many-to-many LSTM	0.1	0.1	0.0
Fully-Connected Autoencoder	0.25	0.35	0.25
LSTM-based Autoencoder	0.15	0.15	0.15
LSTM-based Dual Encoder	0.15	0.15	0.05
Siamese Angular LSTM	0.4	0.35	0.1
Siamese Manhattan LSTM	0.25	0.35	0.1
Re-rank Top 50 (Siamese Angular LSTM)	**0.65**	**0.65**	0.45
Re-rank Top 50 (Siamese Manhattan LSTM)	0.5	**0.65**	0.35
Re-rank Top 100 (Siamese Angular LSTM)	**0.65**	0.45	0.35
Re-rank Top 100 (Siamese Manhattan LSTM)	0.6	0.5	0.35

Table 6.4: Comparison of P@20 for all 3 Use Cases

Individually, the deep learning models presented in Chapter 5 fail to perform on a similar level with BM25. Using a network based on the *Siamese Neural Network* architecture to rerank the top 50 results of BM25, however, achieves top performance for both, *Use Case 1* and *Use Case 2*. *Use Case 3*, on the other hand, presents lower scores across all models. This could be due to the fact that the topic of the underlying thesis is computer verification and model checking, but the vocabulary used in the abstract is very similar to general software engineering and fields related to the automotive industry.

The difference in the presented performance compared to Chapter 5.9 can be caused by several possible reasons. One of them is that our experiments are based on a newly defined ground truth, the supervisor of the original thesis, which might not represent the best possible similarity between abstracts and research profiles. Another reason might be that the selection of use cases is heavily influenced by the background knowledge of the

author of this thesis, resulting in a very small and not very diverse sample size. In order to generate better insights in which model is performing the best, a separate case study of bigger scope should be conducted, i.e. using a bigger pool of unseen abstracts from a variety of different study programmes/topics.

As an additional note, the core focus of *Use Case 1* and *Use Case 2* is on logic and related frameworks, which - as a topic - consists of a very specific vocabulary. Non-relevant documents retrieved in our case study are mostly related to research profiles with a Mathematics background, which shares a significant part of the vocabulary.

CHAPTER 7

Conclusion & Outlook

In the final chapter of this thesis, we present the main outcomes and conclusions based on all experiments we conducted in previous chapters. Therefore, we start by providing (1) a brief summary of the most important results, including potential areas of improvement and concluding statements. Finally, we finish this thesis with (2) an outlook on future work that can be done for the provided context.

7.1 Summary and Conclusions

The main objective of this thesis is to develop a deep learning model which takes the abstract of a thesis as input and returns a ranking list of potential reviewers. As part of this problem, we extracted required data and created our own base data set. Given this data set, we conducted a set of experiments based on different architectures and learning objectives. The purpose of all introduced neural networks can be grouped into two main categories: generating custom embedding vectors and calculating a similarity score.

For the models, which calculate hidden embeddings based on a given document as input, we used the architecture of the base embedding modules BERT, GPT2 and XLNet and presented three additional models. Two of those models process the sentences of the given text corpus as sequence using LSTM layers. For the third model, the averaged sum across all sentence embedding vectors was used to generate a single document embedding, which was then further processed. In Figure 7.1, we see that both LSTM based networks (*Custom Embeddings* and *LSTM AE*) did not manage to outperform the base embeddings of BERT and such. Surprisingly, however, the autoencoder using the single document embedding achieved the best results across all three different embedding modules.

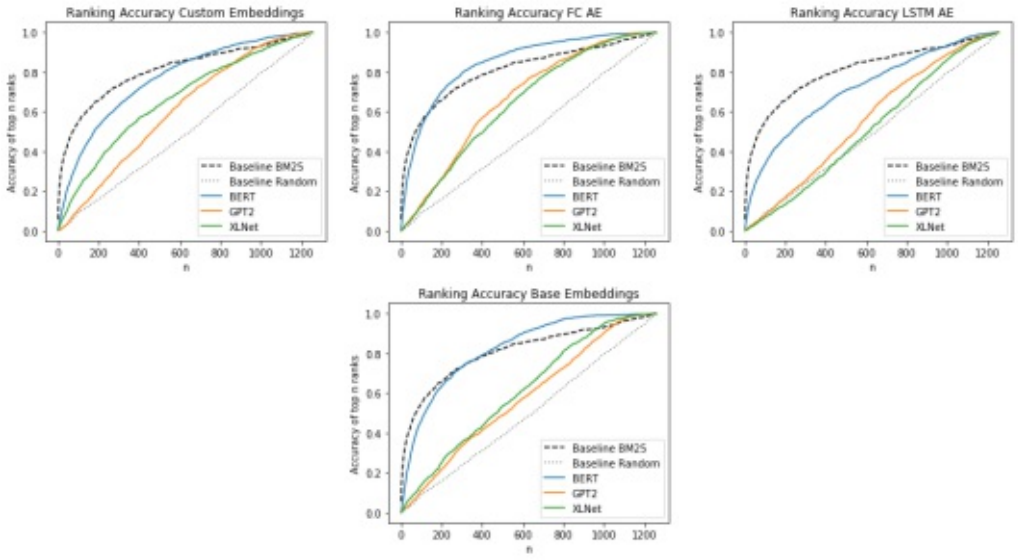

Figure 7.1: Comparison of Custom Embedding generating Models

An even better performance was achieved using the *Siamese Neural Network* inspired architectures, which aim at calculating the similarity score based on two input documents: the abstract of a given thesis and a research profile. In Figure 7.2, we can observe that all three different models outperform our state-of-the-art baseline BM25. For earlier ranks, though, BM25 still remains to be the best performing model. Additionally, we can see that the *Manhattan Similarity* function achieves better results by comparing the Siamese Angular LSTM with the Siamese Manhattan LSTM. Both networks share the exact same architecture with only one difference: the similarity scoring function. Also, it seems that sharing weights across both subnetworks in the siamese network architecture performs slightly better than using two independent subnetworks, which can bee seen in the comparison of the Siamese Angular LSTM with the Dual Encoder.

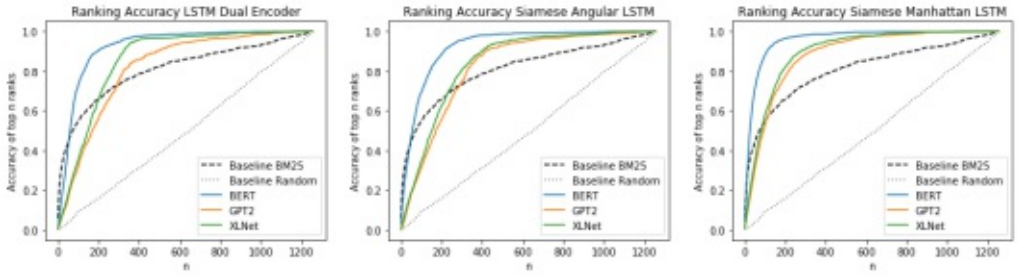

Figure 7.2: Comparison of Siamese Network based Models

However, the performance measured as part of the training process of the different models does not transfer equally to the performance on the main task of this thesis. During the

training process, the aim of the models was to rank the supervisor of a given thesis as high as possible, whereas in the context of this thesis we also desire similar research profiles to be ranked high. In Table 7.1, we can see that although the Siamese Manhattan LSTM clearly outperforms all other networks, including the baseline model, it only achieves a good P@20 score for one of the three use cases presented in Chapter 6. The best scores, excluding the baseline, are highlighed in **bold**, while the lowest scores are marked red.

Model	MRR	UC 1	UC 2	UC 3
Baseline BM25	0.00420	0.5	0.65	0.5
Base Embeddings	0.00458	0.25	0.25	0.2
Many-to-many LSTM	0.00343	0.1	0.1	0.0
Fully-Connected Autoencoder	0.00529	0.25	**0.35**	**0.25**
LSTM-based Autoencoder	0.00284	0.15	0.15	0.15
LSTM-based Dual Encoder	0.01078	0.15	0.15	0.05
Siamese Angular LSTM	0.01098	**0.4**	**0.35**	0.1
Siamese Manhattan LSTM	**0.02005**	0.25	**0.35**	0.1

Table 7.1: Comparison of Model Performance based on MRR and P@20

Overall, we can conclude that for our given learning objective in the training process, the *Siamese Neural Network* architecture shows promising results. The Siamese Manhattan LSTM even manages to outperform the BM25 baseline starting with a rank of 10. We assume that with enough fine-tuning of the model, including the architecture, it is possible to achieve better results across all ranks. However, in the setting of ranking similar research profiles as potential reviewers, the BM25 baseline retains the best performance. As presented in Chapter 6.5, we can achieve similar performance by employing a re-ranking strategy, i.e. using BM25 for the initial ranking list and, afterwards, re-rank the top 50 results with the Siamese Angular LSTM (see Table 7.2 for reference).

As the BM25 baseline was not trained with the theses supervisors as target, we assume that one reason for the discrepancy in the evaluation metrics is the quality of our base data set. Research profiles for a given target reviewer might not be complete or lack information related to the underlying thesis content. This will cause the models to learn patterns of similar documents, which show similarly poor data quality, resulting in a bad performance for the problem setting of this thesis. Additionally, the assignment of target reviewers using the initial supervisors can potentially lead to poor reviewer-thesis pairs, which are the main input for our neural networks to learn.

Model	UC 1	UC 2	UC 3
Baseline BM25	0.5	**0.65**	**0.5**
Re-rank Top 50 (Siamese Angular LSTM)	**0.65**	**0.65**	0.45
Re-rank Top 50 (Siamese Manhattan LSTM)	0.5	**0.65**	0.35
Re-rank Top 100 (Siamese Angular LSTM)	**0.65**	0.45	0.35
Re-rank Top 100 (Siamese Manhattan LSTM)	0.6	0.5	0.35

Table 7.2: Evaluation of the Re-ranking Strategy based on the top performing Models

7.2 Outlook

This leads us to the outlook on future work and possible improvements, where we mainly cover three different areas: (1) the data set, (2) the network architecture, and (3) the problem setting.

Data Set. A potential area for future work is the creation of a new data set. Especially the task of generating meaningful reviewer-thesis pairs seems promising as this dictates the performance of our models, regardless of the chosen architecture. A well balanced and diverse data set could also further improve the results of the already presented architectures. Therefore, we suggest the creation of a bigger data set, including additional positive and - more importantly - negative samples for each of the reviewers, where the definition of a novel negative sampling method in the given context can be considered another area of improvement.

Network Architecture. In the scope of this thesis, we mainly focused on the generation of custom embeddings using encoder-decoder architectures and similarity scoring based on *Siamese Neural Networks*. For future work, we could search for other suitable types of architectures or try to improve existing approaches by further fine-tuning its hyperparameters such as the layers. In addition to that, we could also explore different learning approaches such as semi-supervised learning and active learning.

Problem Setting. Finally, we suggest to extend the work presented in this thesis by looking at different problem settings or expand the context of application. For a different problem setting, we would take the same architectures as introduced in the main part of this thesis and train them on different data sets such as citation networks, i.e. using connected papers as target values for similar documents. The resulting models can then be evaluated in the context of this thesis. Another possibility is to use the already trained models and measure their performance in related problem settings such as the selection process of reviewers for paper submissions at conferences.

List of Figures

List of Tables

Bibliography

[AGB+19] Omer Anjum, Hongyu Gong, Suma Bhat, Wen-Mei Hwu, and JinJun Xiong. PaRe: A paper-reviewer matching approach using a common topic space. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 518–528, Hong Kong, China, November 2019. Association for Computational Linguistics.

[ARTL19] Ashutosh Adhikari, Achyudh Ram, Raphael Tang, and Jimmy Lin. DocBERT: BERT for Document Classification. *arXiv e-prints*, page arXiv:1904.08398, April 2019.

[AWN18] Hosameldin Ahmed, M. Wong, and Asoke Nandi. Intelligent condition monitoring method for bearing faults from highly compressed measurements using sparse over-complete features. *Mechanical Systems and Signal Processing*, 99:459–477, 01 2018.

[Ber20] Michael Bernreiter. A general framework for choice logics. Master's thesis, TU Wien, 2020.

[BKL09] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[BRN+09] Dominic Balasuriya, Nicky Ringland, Joel Nothman, Tara Murphy, and James R. Curran. Named entity recognition in wikipedia. In *Proceedings of the 2009 Workshop on The People's Web Meets NLP: Collaboratively Constructed Semantic Resources*, People's Web '09, page 10–18, USA, 2009. Association for Computational Linguistics.

[Bro20] Dorian Brown. Rank-bm25: A two line search engine. `https://github.com/dorianbrown/rank_bm25`, 2020. last visited on 10-01-2021.

[BS20] Ursin Brunner and Kurt Stockinger. Entity matching with transformer architectures - A step forward in data integration. In Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan

Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang, editors, *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, pages 463–473. OpenProceedings.org, 2020.

[CDOK20] Stephan A. Curiskis, Barry Drake, Thomas R. Osborn, and Paul J. Kennedy. An evaluation of document clustering and topic modelling in two online social networks: Twitter and reddit. *Information Processing & Management*, 57(2):102034, 2020.

[CGCB14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv e-prints*, page arXiv:1412.3555, December 2014.

[Cra09] Nick Craswell. *Mean Reciprocal Rank*, pages 1703–1703. Springer US, Boston, MA, 2009.

[CYK+18] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Universal Sentence Encoder. *arXiv e-prints*, page arXiv:1803.11175, March 2018.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[DLLK18] G. Dong, G. Liao, H. Liu, and G. Kuang. A review of the autoencoder and its variants: A comparative perspective from target recognition in synthetic-aperture radar images. *IEEE Geoscience and Remote Sensing Magazine*, 6(3):44–68, 2018.

[Dur20] Timothée Durand. Model checking automotive software components. Master's thesis, TU Wien, 2020.

[Han18] SuHun Han. Googletrans. `https://github.com/ssut/py-googletrans`, 2018. last visited on 10-01-2021.

[Hof19] Sebastian Hofstätter. Lecture: Advanced information retrieval, 2019.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[Hun07] John D. Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[Kar15] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. 2015. last visited on 06-01-2021.

[Koc15] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015.

[KSMM20] Divya Khyani, Siddhartha B S, Niveditha N M, and Divya B M. An interpretation of lemmatization and stemming in natural language processing. *Journal of University of Shanghai for Science and Technology*, 22(10):350–357, October 2020.

[Kö20] Matthias König. Graph-classes of argumentation frameworks with collective attacks. Master's thesis, TU Wien, 2020.

[MC17] Bhaskar Mitra and Nick Craswell. Neural Models for Information Retrieval. *arXiv e-prints*, page arXiv:1705.01509, May 2017.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119. Curran Associates, Inc., 2013.

[Nav09] Roberto Navigli. Word sense disambiguation: A survey. *ACM computing surveys (CSUR)*, 41(2):1–69, 2009.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Ram03] Juan Ramos. Using tf-idf to determine word relevance in document queries. 2003.

[RG19] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv e-prints*, page arXiv:1908.10084, August 2019.

[Ric07] Leonard Richardson. Beautiful soup documentation. *April*, 2007.

[RMPA16] Nihar Ranjan, Kaushal Mundada, Kunal Phaltane, and Saim Ahmad. Article: A survey on techniques in nlp. *International Journal of Computer Applications*, 134(8):6–9, January 2016. Published by Foundation of Computer Science (FCS), NY, USA.

[RMvdW+20] Charles R., K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Harris, array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[RWA+19] Alec Radford, Jeffrey Wu, Dario Amodei, Daniela Amodei, Jack Clark, Miles Brundage, and Ilya Sutskever. Bm25 the next generation of lucene relevance. `https://openai.com/blog/better-language-models/#sample1`, February 2019. last visited on 18-01-2021.

[RWC+19] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[RWJ+94] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at trec-3. In Donna K. Harman, editor, *TREC*, volume 500-225 of *NIST Special Publication*, pages 109–126. National Institute of Standards and Technology (NIST), 1994.

[SW10] Claude Sammut and Geoffrey I. Webb, editors. *Mean Squared Error*, pages 653–653. Springer US, Boston, MA, 2010.

[Thy15] Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. 11 2015.

[Tur15] Doug Turnbull. Bm25 the next generation of lucene relevance. `https://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/`, October 2015. last visited on 17-01-2021.

[VK16] M.K. Vijaymeena and K. Kavitha. A survey on similarity measures in text mining. *Machine Learning and Applications: An International Journal*, 3(2):19–28, 2016.

[VRD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017.

[WDS+20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[YDY+19] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 5753–5763. Curran Associates, Inc., 2019.

[Zei12] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv e-prints*, page arXiv:1212.5701, December 2012.

[ZZD+20] Dong Zhang, Shu Zhao, Zhen Duan, Jie Chen, Yanping Zhang, and Jie Tang. A multi-label classification method using a hierarchical and transparent representation for paper-reviewer recommendation. *ACM Trans. Inf. Syst.*, 38(1), February 2020.