Eötvös Loránd University
Faculty of Informatics

Application of transformers in scientific literature mining

Dr. Horváth Tamás
Head of Department of Data Science and Engineering at ELTE
Dr. Jean Martinet
Full Professor at Université Côte d'Azur
Dr. Alexey Isavnin
Co-founder at Rays of Space Oy

Kseniia Isavnina
Computer Science

Budapest, 2023.

Acknowledgement

We would like to express my appreciation to Dr. Alexey Isavnin and Dr. Konstantin Avchaciov for brilliant ideas on a way of how to approach this challenging topic.

Special thanks to Dr. Horváth Tamás for support, constructive remarks, and empathy.

We would also like to thank Dr. Jean Martinet for feedback, and Várhalmi Zsuzsi and Katalin Schneider for administrative support so crucial for researchers.

Contents

Chapter 1

Abstract

This work presents a project focused on the application of transformers and attention mechanisms in Natural Language Processing (NLP) to generate citation suggestions in the scientific research domain. The main objective is to explore the utilization of these techniques and their potential impact on NLP tasks. The study begins by investigating transformers and attention mechanisms, understanding their relevance in NLP. Specifically, the architecture of SciBERT, a pre-trained language model, is examined to analyze how it captures and represents dependencies within text.

To facilitate the generation of citation suggestions, a dataset comprising scientific papers and their associated citations is acquired. The dependencies within the citations are extracted using open-source sources such as Semantic Scholar. Transfer learning with the aid of contrastive learning is then applied to fine-tune the SciBERT model with the aim of improving the quality and relevance of the generated citation suggestions.

Experimental evaluations are conducted to assess the effectiveness of the fine-tuned model. The experiment performed on a selected paper reveals that the true references, along with other recommendations obtained from comparing the embedding vectors, are slightly more densely distributed towards the top recommendations compared to the overall average.

Moving forward, retraining SciBERT with contrastive loss becomes a crucial next step. By unfreezing the weights of the underlying SciBERT model and increasing the training dataset size, it is expected that the citation suggestion quality can be further enhanced. This approach allows for better incorporation of domain-specific knowledge and fine-tuning the model specifically for the citation suggestion task. Additionally, the previously mentioned steps, such as expanding the dataset to include a more diverse range of scientific disciplines and addressing the input size constraints for texts, should also be pursued. These steps aim to improve the fairness, inclusivity, and overall performance of the citation suggestion system. By following this step-by-step approach and giving due importance to retraining SciBERT with contrastive loss, this project aims to advance the generation of citation suggestions in scientific research by leveraging transformers and attention mechanisms in NLP.

Chapter 2

Introduction

The objective of the study is to investigate the possibilities of transformers in scientific literature mining. In practice that means the system should provide the list of references to cite based on the written text. In order to narrow the task, we took an abstract of a paper as an example of this "written text".

We are aiming to provide researchers with the writing and research assistant capable of suggesting the resources helpful in their research. We want to implement text analysis that will analyze the content written by the user, make an assumption of the related research based on a semantic distance to the existing research in the field, and suggest the research material to cite or to add to the reading list. We assume that this feature could speed up the process of research and increase the collaboration between the researchers in the same or related fields of study.

We are aiming to implement state-of-the-art research in Natural Language Processing (NLP). We have chosen transformers for this take for a number of reasons. The ability of transformers to handle sequential data, model long-term dependencies, and achieve state-of-the-art performance has made them a popular choice for NLP tasks.

First of all, we will need to handle sequential data meaning text, where the order of the words is important, and transformers are specifically designed to handle sequential data by processing the entire sequence of input tokens at once, unlike traditional recurrent neural networks that process input tokens one at a time. Secondly, we will need to model long-term dependencies.

Transformers, thanks to the self-attention mechanism, can relate every input token to every other input token, allowing them to better model long-term dependencies. And finally, due to the lack of resources and limitation in time it sounds reasonable to focus on pretrained models in NLP, such as BERT [DCT19], for example.

By researching transformers in NLP, we can build on this foundation and explore new directions for language modeling and natural language understanding.

Chapter 3

Project Description

3.1 Technological Context

First of all, we would like to narrow our task to a specific context in order to increase the possibility of a successful delivery of our proof of concept. We have chosen a scientific context, specifically focusing on the computer science field of study, which aligns closely with our area of expertise, and which also could intersect with the other cross-disciplinary research.

Our goal is to develop a system that takes this text in as input and provides a list of related papers that can be cited. To achieve this, we will leverage the capabilities of a neural network trained to understand scientific texts and extract relevant features that can be used as word embeddings for classification. One such neural network model that we can utilize is SciBERT [BLC19], which has been specifically trained on scientific literature.

In addition to the neural network model, we need to create a hyperspace of closely related papers. This process can be likened to clustering, where we aim to map the resulting feature output to relevant papers. To create this hyperspace, we will use the citations of scientific papers as anchor points. By estimating the distance between papers based on their citations, we can establish the proximity of related papers.

To train our final layers of the SciBERT model for this classification task, we will utilize a loss function. This function plays a crucial role in guiding the model's learning process through unsupervised learning technique. Some of the loss functions being used for such kind of task include:

Cross-Entropy Loss [ZS18]: This loss function is commonly used for classification tasks and encourages the model to minimize the difference between predicted and actual class labels.

Triplet Loss [HA14]: The triplet loss aims to minimize the distance between similar texts while maximizing the distance between dissimilar texts. It helps the model learn embeddings that can effectively separate different classes.

Margin Loss: Similar to triplet loss, margin loss focuses on maximizing the margin between similar texts and dissimilar texts. By increasing the margin, the model can achieve better separation between classes. This loss was widely used in face recognition tasks, however, masked multi-center angular margin loss was also used for the for language recognition task [JXKS22].

Margin Cosine Loss (or CosFace) [WWe18]: This loss function adds an angular margin to the cosine similarity between features and class centers. It encourages better separation between different classes by increasing the angular distance.

Center Loss [QS17]: The center loss is used to learn discriminative features by minimizing the distance between features and their corresponding class centers. It helps in capturing class-specific characteristics. We consider this approach to be a perspective one for our task as well, and it will make sense to compare its effectiveness with ArcFace in the future.

ArcFace [DGZ19]: Initially designed for face recognition tasks, a type of margin loss, ArcFace introduces an angular margin to the softmax loss. By applying this concept to our NLP task, we can potentially enhance the recognition of related papers by introducing better separation between classes.

We will experiment with ArcFace. Our intuition is that by treating words as entities rather than pixels, we can leverage the concept of ArcFace and apply it to our "paper recognition" task. This approach, combined with classification based on multidimensional feature learning, can capture the intricate dependencies between words in a scientific document.

Through our research and experimentation, we aim to enhance the quality of our dataset and develop a robust system for generating abstracts and suggesting relevant citations for scientific papers.

3.2 Project Downstream

In this project, our focus is on understanding and applying transformers and attention mechanisms to natural language processing (NLP) tasks. We will delve into the architecture of SciBERT, examining its layers and exploring how we can extract features that capture the dependencies present in text.

To kickstart our project, we need a dataset of papers and their citations. We will explore open-source sources such as Semantic Scholar to uncover the possibilities of extracting dependencies within citations. By leveraging these resources, we can enhance our understanding of the relationships and connections between scientific papers.

Once we have gathered the necessary data, we will proceed with transfer learning on SciBERT. Transfer learning allows us to leverage the pre-trained knowledge of SciBERT and fine-tune it for our specific task. By adapting the model to our dataset, we can enhance its performance and make it more effective in generating meaningful suggestions for citations.

Here is the downstream of the whole process:

- Explore Transformers and Attention Mechanisms:
  - Learn about the fundamentals of transformers and attention mechanisms in the context of NLP.
  - Understand how these techniques can improve various NLP tasks, including citation suggestion.
- Investigate SciBERT Architecture:
  - Dive into the architecture of SciBERT, examining its layers and components.
  - Identify how SciBERT captures and represents dependencies within text through its feature extraction process.
- Acquire Dataset of Papers and Citations:
  - Source a suitable dataset of scientific papers and their associated citations.
  - Consider utilizing open-source platforms like Semantic Scholar [**?**] to access relevant data.
- Extract Dependencies within Citations:
  - Explore the possibilities of extracting dependencies within citations from the acquired dataset.
  - Utilize techniques and tools to analyze and capture meaningful connections between cited paper.
- Perform Transfer Learning [RS19] on SciBERT:
  - Apply transfer learning techniques to fine-tune the pre-trained SciBERT model.
  - Adapt SciBERT to the specific task of generating citation suggestions based on the acquired dataset.

- Run Experiments:
    - Execute experiments using the trained SciBERT model and the dataset.
    - Generate citation suggestions and evaluate their quality and relevance.

Through this downstream project, we aim to gain a deeper understanding of how transformers and attention mechanisms can be effectively applied in NLP tasks. By exploring the architecture of SciBERT, extracting dependencies from citations, conducting transfer learning, and analyzing the performance of the algorithm, we strive to find a way to enhance the quality and effectiveness of citation suggestions.

With the model trained and fine-tuned, we will run experiments to evaluate its performance. It is worth analyzing the quality of the citation suggestions by seeking the human feedback. This is an iterative process which involves also usability tests, and agile approach to software development which will influence modelling as well so that we can obtain valuable insights into the accuracy and relevance of the generated suggestions. However, this last step is mostly outside of the scope of the current thesis work.

Chapter 4

Process Of Work And Methodology

4.1 Model Selection: Leveraging SciBERT for Text Embedding and Comparison

After several discussions with Dr. Alexey Isavnin and Dr. Konstantin Avchaciov, relevant professionals in the field, about the possibilities of the proposed approach, we came to the conclusion the pre-trained language model SciBERT, would possibly fit for the task. SciBERT is a variant of BERT that has been specifically pre-trained on a random sample of the full text of 1.14M papers from Semantic Scholar from Biomedical and Computer Science domains. The proportion of vocabulary is shown at Table 4.1. 80% of words are considered to be from the Biomedical domain while 20% of words came from the Computer Science domain. The average paper length is 154 sentences (2,769 tokens) resulting in a corpus size of 3.17B tokens, similar to the 3.3B tokens on which BERT was trained .

Table 4.1: The proportion of vocabulary used for BERT training, and SciBERT training.

	Corpus	# of words
BERT	English Wiki	2.5B
	BooksCorpus	0.8B
SciBERT	Biomedical	2.5B
	Computer Science	0.6B

As a result, it has been shown to perform well on a wide range of natural language processing tasks in the scientific domain, including text classification, named entity recognition, and relation extraction. It is worth mentioning that SciBERT has its own biases and constraints due to its specific design to handle and comprehend scientific text, particularly in computer science and biomedical domains. This training corpus encompasses a wide range of subfields, including artificial intelligence, software engineering, bioinformatics, genetics, and more. By incorporating a diverse array of topics, SciBERT gains exposure to the language patterns, terminology, and research context prevalent in these domains.

As far as our task is concerned, we understand that extracting valuable information from scientific text can be challenging due to its complex language, domain-specific terminology, and diverse document structures. Due to the fact that SciBERT addresses these challenges showing the best performance in computer science and biomedical domains (as it was trained on them), it will probably make sense to run our experiment on a similar type of dataset, and evaluate the model also on a scientific paper from those domains. SciBERT's training on a corpus of articles in computer science and medicine makes it particularly suitable for generating embedding vectors for scientific texts in these specific domains.

We are going to investigate the extent to which SciBERT can generate feature embedding vectors for scientific texts. By comprehending the strengths and limitations of SciBERT, we aim to harness its capabilities effectively to enhance a variety of natural language processing tasks within computer science and biomedicine.

We had an intuition that SciBERT might be well-suited for comparing embedding vectors because it has been trained on a large corpus of scientific text, which is likely to contain a diverse range of vocabulary and complex syntax and probably able to capture the nuances of scientific language and produce high-quality embeddings, necessary to cluster the papers together with the help of the loss function that we choose. The lower the value of the loss function is, the closer papers to each other are in the semantic hyperspace.

Another reason for choosing SciBERT is that it uses a transformer architecture, which is particularly effective at modeling long-range dependencies in text. This means that the model is able to capture the context of each word in a sentence, even if the words are separated by a long distance. This could be beneficial for comparing embeddings, as it allows the model to consider the relationship between each pair of words in the sentence.

4.2 Embedding Vectors for Text Similarity Analysis: Advantages and Alternatives

The reasonable question would be why did we decide to use embedding vector analysis generated by SciBERT for our text similarity task. There are multiple other ways to perform it. These methods vary in their approach and can be used depending on the specific requirements and characteristics of the text data. For example:

- String-Based Similarities [GF13] based on the analysis of sequences of characters or token sequences like Jaccard similarity, Levenshtein distance, Cosine similarity, and Edit distance. These metrics consider the textual overlap, character-level differences, or token-level similarities between texts.
- The other popular method is TF-IDF [Ram03] (Term Frequency-Inverse Document Frequency): TF-IDF calculates the importance of words in a document by considering their frequency in the document relative to their frequency in the entire corpus. Texts can be compared using the cosine similarity of their TF-IDF vectors. This approach highlights the distinguishing terms in each document and enables similarity analysis based on term importance.
- Another approach is Word Overlap. It compares texts based on the number of overlapping words or tokens they share. While straightforward, this method does not consider the semantic meaning or context of the words.
- N-gram Similarity [Kon05]: N-grams are contiguous sequences of n items (characters, words, etc.) within a text. N-gram similarity measures compare the presence and frequency of n-grams between texts. By comparing the overlap of n-grams, such as unigrams, bigrams, or trigrams, text similarity can be assessed based on shared sequences of words or characters.
- Topic Modeling: Topic modeling techniques, such as Latent Semantic Analysis (LSA) or Latent Dirichlet Allocation (LDA) [JWF18], identify latent topics within a corpus. Texts can be compared based on their distribution of topics or the similarity of their topic proportions. This approach enables similarity analysis based on thematic similarities between texts.
- Graph-Based Methods [PRZS16]: Graph-based methods represent texts as nodes in a graph, where edges capture relationships between texts. Similarity can be measured using graph-based algorithms, such as graph similarity algorithms or graph kernels. This approach considers the connectivity and structural similarities between texts within a graph representation. Which we do not have, or have limitations to access it.
- Deep Learning-Based Approaches: Siamese networks[RG19] or attention-based models (like BERT) are commonly used for text similarity tasks. These models learn complex representations and capture contextual information to assess the similarity between texts.

- Ensemble Approaches: Ensemble methods combine multiple similarity measures or models to improve the robustness and accuracy of text similarity analysis. By leveraging the strengths of different methods, ensemble approaches can provide more reliable similarity scores or rankings.

All of these approaches are viable and can perform well depending on a specific task, the nature of the text data, the available resources, and the desired level of granularity in the similarity assessment. It is beneficial to experiment with multiple methods and evaluate their performance against the specific requirements of the task at hand.

Among the various approaches to text similarity, deep learning-based methods, particularly those utilizing transformer-based models like BERT (or SciBERT in our case), have gained significant traction in recent years. These models have become increasingly popular due to their ability to capture contextual information, handle complex semantic relationships, and achieve state-of-the-art performance on various natural language processing tasks.

The rise in popularity of deep learning-based methods for text similarity can be attributed to several factors like contextual understanding, enabling them to generate more nuanced and precise representations of text, leading to improved similarity assessments. Those models are pre-trained on large-scale corpora, such as Wikipedia or the entire internet (Semantic Scholar database in our case).

It is important to mention that deep learning models, particularly pre trained ones, have strong transfer learning capabilities. This transfer learning paradigm facilitates the application of pretrained models to text similarity tasks, eliminating the need for extensive task-specific labeled data and accelerating development. We will use this possibility also due to the lack of computational sources. Another tempting feature of such an approach is the availability of those pre-trained models meaning the democratized access to advanced text similarity capabilities. These models can be readily used in such tasks as ours, the architecture can be slightly modified, and the hyperparameters fine-tuned according to the requirements of the specific similarity task with relatively minimal effort. Deep learning models, especially transformer-based architectures, have achieved state-of-the-art performance on various text similarity benchmarks and tasks. Their ability to capture intricate semantic relationships, understand context, and leverage large-scale pretrained representations has significantly advanced the field.

While deep learning-based approaches, especially transformer-based models, are currently trending in text similarity research, it's important to note depending on the dataset, computational resources, interpretability needs, and other factors, alternative methods such as string-based metrics, TF-IDF, or graph-based approaches may still be viable and effective options, and may be added to the chosen method - embedding vector comparison.

Comparing embedding vectors for text similarity offers several advantages over traditional methods of text comparison. Embedding models like BERT or SciBERT leverage contextual information to generate embeddings. This contextual understanding enables the model to capture the meaning of words based on their surrounding context, considering both direct and indirect relationships. Consequently, comparing embedding vectors considers the semantic nuances present in the text, leading to more accurate similarity assessments.

Embedding models can be trained in an unsupervised manner, leveraging large corpora without explicit annotations, which is exactly our case. This makes them applicable to a wide range of texts, including those without labeled data. By extracting embeddings and comparing them, text similarity can be assessed without the need for extensive labeled datasets, making the process more accessible, cost-effective, and less biased. For example, in our case, we want to learn the loss function in order to identify a union or papers, which are semantically closer to each other, and create the label for them.

Embedding vectors represent text in lower-dimensional vector spaces. This dimensionality reduction not only simplifies the comparison process but also reduces computational complexity. As a result, comparing embedding vectors can be more efficient and scalable, particularly when dealing with large datasets.

Embedding models are trained on large-scale datasets, providing a general understanding of language. We assume that this property will allow us to compare the vectors from different languages as the consequence of the property of the learned embeddings to generalize well the unseen or out-of-domain text.

This gives us the possibility to make a comparison of texts across different domains as well, facilitating tasks such as cross-domain similarity analysis or multilingual text comparison. But this task is beyond current experiment and will require more resources. So, we will stick to our chosen domains for now, and one chosen language - English.

4.3 Unsupervised Learning for Text Similarity and Label Generation

Therefore, in our study, we acknowledge that text similarity is a multifaceted problem that necessitates a comprehensive understanding of the task and familiarity with the underlying architecture of embedding models. However, prior to delving into this intricate task, it is imperative to address the crucial issue of data labeling. In the context of text similarity, the labels signify the grouping of papers based on their predefined degrees of similarity.

Ensuring the generation of unbiased labels is paramount in establishing a dependable and accurate system. One plausible approach is to construct a classification system by leveraging an existing taxonomy such as Wikipedia. Nonetheless, it is essential to recognize that this method has its limitations, as it can still introduce biases inherent in the chosen classification.

Alternatively, we propose an alternative approach that involves training a loss function and utilizing unsupervised learning techniques to generate labels during the training process. By doing so, we can mathematically define the dependencies between texts and identify clusters of related papers. The underlying assumption is that a paper, along with the set of papers it cites, as well as those that cite the references within the initial paper, collectively form a cohesive cluster.

This concept aligns with the notion of a citing hyperspace, where papers are interconnected through citations, forming a network of knowledge dissemination (refer to 4.1). Certain papers receive frequent citations from diverse topics, acting as bridges that connect different areas of knowledge. By learning an appropriate loss function, we can effectively identify these clusters and uncover similarities between texts.

The ultimate objective of our research is to apply this approach to scientific literature, leveraging citation patterns to unveil the intricate connections between cross-domain papers. By employing our proposed method, we aim to shed light on the underlying relationships and possible scalability of that approach across a wide range of scientific disciplines. This has the potential to enhance our understanding of the cumulative knowledge within the scientific community and facilitate cross-disciplinary collaboration.

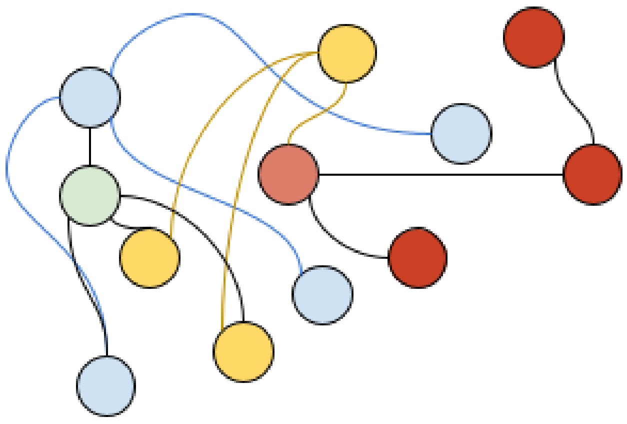


Figure 4.1: Abstract representation of papers citing each other.png

Unsupervised learning approaches, such as clustering and self-supervised learning, have shown promising results in identifying similarities between texts without the need for explicit labels [GNW21]. One popular approach is contrastive learning, where the model learns to distinguish between similar and dissimilar pairs of texts. Recent research, such as the paper "SimCSE: Simple Contrastive Learning of Sentence Embeddings" by Gao et al. (2021) [GYC21], has shown that contrastive learning can significantly improve the performance of models on downstream natural language understanding tasks compared to other unsupervised learning approaches.

The benefit of using deep learning-based approaches is that they can learn to recognize patterns and relationships that might not be immediately apparent to human analysts. This can be especially helpful in the scientific domain, where papers may use different terminology but still address similar research questions.

By using deep unsupervised learning approaches, we can generate the labels needed for training from the data itself, without the need for human-labeled data. This makes this approach more accessible to researchers in other domains who may not have access to labeled data and unlock new insights and patterns in text data that may have been previously undiscovered.

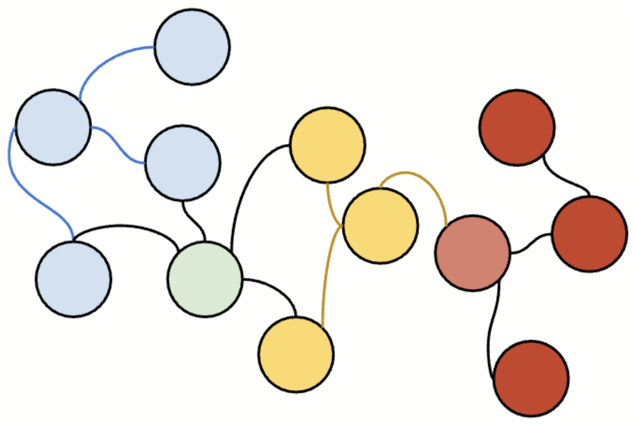


Figure 4.2: Abstract representation of papers in a citing hyperspace after training the loss function.

By minimizing the loss function, the model learns to map abstracts into an embedding space in which similar abstracts are mapped to nearby points and dissimilar ones are mapped to distant points. This enables grouping or clustering of papers based on their semantic similarity. (Figure 4.2)

4.4 Contrastive Learning: Exploring the Loss Function and Calculation Process

The basic idea of contrastive learning is to encourage the model to map similar examples to nearby points in the representation space while pushing dissimilar examples further apart. This can be achieved by training the model to discriminate between pairs of examples in a way that maximizes the agreement between similar pairs and minimizes the agreement between dissimilar pairs.

The idea of contrastive learning has been around for several decades and has been used in various domains, such as computer vision, speech recognition, and natural language processing. In the context of language modeling, contrastive learning has been used to learn sentence embeddings, which are fixed-length representations of text that capture their meaning and semantics.

In recent years, contrastive learning has become increasingly popular in the field of deep learning, particularly in the context of self-supervised learning. This is because it can be used to learn effective representations without the need for explicit supervision, which is often expensive and time-consuming to obtain.

The main benefit of contrastive learning is that it allows us to learn rich, informative representations of data that can be used for a wide range of downstream tasks. By leveraging the contrastive loss, we can learn to distinguish between similar and dissimilar examples, even in the absence of explicit labels.

Contrastive loss is negative log likelihood of the true class label y under the softmax function. It is computed by dividing the exponential of the modified angle by the sum of exponential terms across all class labels and taking the negative logarithm.

The formula for the ArcFace contrastive loss [DGZ19]:

$$L = -\log\left(\frac{\exp\left(s \cdot \cos\left(\theta_y + m\right)\right)}{\exp\left(s \cdot \cos\left(\theta_y + m\right)\right) + \sum_{j \neq y} \exp\left(s \cdot \cos\left(\theta_j\right)\right)}\right) \tag{4.1}$$

where:

θ_y - is the angle between the input feature vector and the weight vector corresponding to the true class label. It is computed using the arccosine of the dot product between these two vectors:

$$\theta_y = \arccos\left(\frac{\mathbf{w}_y^T \mathbf{x}}{\|\mathbf{w}_y\| \, \|\mathbf{x}\|}\right) \tag{4.2}$$

where:

$\mathbf{x}$ is the input feature vector, $\mathbf{w}_y$ is the weight vector corresponding to the true class label y.

θ_j - is the angle between the input vector and the weight vector of the other classes,

$\cos\theta_j$ - is the cosine of the angle between the input feature vector and the weight vector corresponding to each class label j. It is simply the dot product between these two vectors normalized by their Euclidean norms.

$$\cos\theta_j = \left(\frac{\mathbf{w}_j^T\mathbf{x}}{\|\mathbf{w}_j\|\,\|\mathbf{x}\|}\right) \tag{4.3}$$

,

s - is a scaling factor,

m - is the angular margin.

The contrastive loss is based on traditional softmax function 4.7. Traditional softmax loss is widely used, for example, in deep face recognition. However, the softmax loss function does not explicitly optimize the feature embedding to enforce higher similarity for intra-class samples and diversity for inter-class samples, which is required for our text similarity analysis. The formula for the softmax loss, is presented as follows:

$$L_1 = -\log\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{N} e^{W_j^T x_i + b_j}}, \tag{4.4}$$

where $x_i \in \mathbb{R}^d$ denotes the deep feature of the i-th sample belonging to the y_i-th class. The embedding feature dimension d is set to 512 due to the nature of the SciBERT model. $W_j \in \mathbb{R}^d$ denotes the j-th column of the weight $W \in \mathbb{R}^{d\times N}$, $b_j \in \mathbb{R}^N$ is the bias term, and the class number is N.

Then, we transform the logit $W_j^T \cdot x_i = ||W_j|| \cdot ||x_i|| \cdot \cos(\theta_j)$, where θ_j is the angle between the weight W_j and the feature x_i. We also fix the individual weight $k_{W_j} = 1$ by L_2 normalization.

The contractive loss term 4.1 encourages the cosine similarity between the input feature vector and the weight vector corresponding to the true class label to be larger than the cosine similarity between the input feature vector and the weight vectors corresponding to the other class labels by a margin of m.

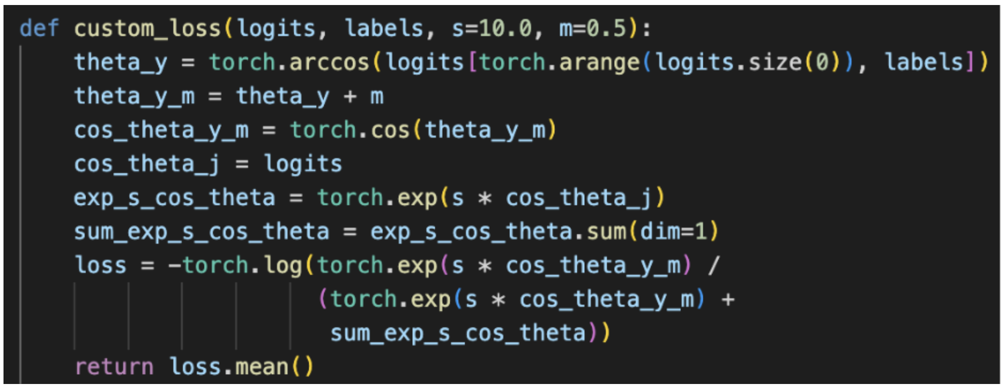
The scaling factor s controls the spread of the softmax function 4.7. A larger value of s leads to a sharper softmax function, making the model more confident in its predictions.

The normalization step on features and weights makes the predictions only depend on the angle between the feature and the weight. The learned embedding features are thus distributed on a hypersphere with a radius of s.

The final loss term, which is negative log likelihood of the true class label y under the softmax function. It is computed by dividing the exponential of the modified angle by the sum of exponential terms across all class labels and taking the negative logarithm.

The formula calculates the loss L for a given input and target label. When learning the loss function for simplicity we fix the bias as zero.

The code for the loss function looks like this:



```python
def custom_loss(logits, labels, s=10.0, m=0.5):
    theta_y = torch.arccos(logits[torch.arange(logits.size(0)), labels])
    theta_y_m = theta_y + m
    cos_theta_y_m = torch.cos(theta_y_m)
    cos_theta_j = logits
    exp_s_cos_theta = torch.exp(s * cos_theta_j)
    sum_exp_s_cos_theta = exp_s_cos_theta.sum(dim=1)
    loss = -torch.log(torch.exp(s * cos_theta_y_m) /
                      (torch.exp(s * cos_theta_y_m) +
                       sum_exp_s_cos_theta))
    return loss.mean()
```

Figure 4.3: The contrastive loss in code

To sum up, in order to calculate the loss function we need to:

1. Calculate the arccos of the predicted logits for the true labels to get the angle between the predicted embedding and the class embedding of the true label.
2. Add the margin parameter m to this angle to encourage greater separation between different classes.
3. Convert the new angle back to a cosine value using the cos function.
4. Calculate the cosine values of all the logits.
5. Scale the logits by the hyperparameter s, which controls the temperature of the softmax function.
6. Exponentiate and sum the scaled logits over all classes to get the denominator of the final loss calculation.
7. Calculate the negative logarithm of the ratio of the exponentiated, scaled cosine of the angle between the predicted embedding and the true class embedding to the sum of the exponentiated, scaled logits for all classes. This is the final loss.
8. Return the mean of all losses as the overall loss for the batch.

4.5 Understanding BERT, Feature Embedding, and MyModel Architecture

We already mentioned that SciBERT is based on BERT that has been specifically pre-trained on scientific literature. BERT [DCT19] (Bidirectional Encoder Representations from Transformers) consists of a stack of transformer layers.The transformer architecture is a key component of BERT and is composed of two main parts: the encoder and the decoder. However, BERT only uses the encoder part (see Figure 4.6), as it is primarily focused on pre-training for language understanding tasks. The encoder part of BERT comprises a stack of identical transformer layers. Each transformer layer consists of two sub-layers: the self-attention mechanism and the position-wise fully connected feed-forward network.

The self-attention mechanism allows BERT to capture the contextual relationships between words or tokens in a given text. It computes attention weights for each word in the input sequence, considering the dependencies and relationships between all other words. The illustration of the attention mechanism is shown in Figure 4.4.

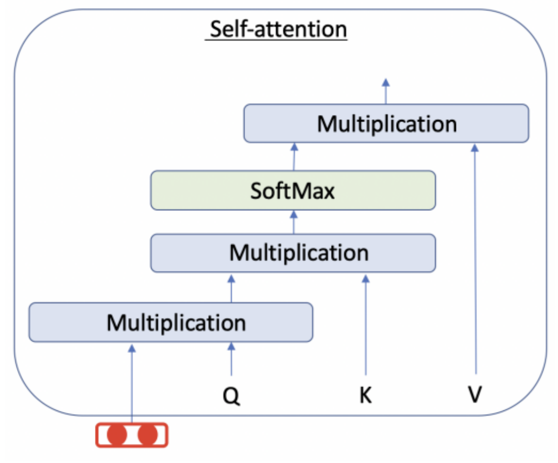


Figure 4.4: An illustration of the Self-Attention mechanism in the Transformer Encoder

A Self-Attention layer maps the input embeddings, a query (Q) and a set of key-value pairs (K, V) to an output vector using an attention mechanism (see Equation 4.5). Given the query and key vectors of dimension d_k, as well as value vectors of dimension d_v, the attention score is calculated as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK}{\sqrt{d_k}}\right)V \tag{4.5}$$

Here, the layer computes the dot products of the query (Q) with all keys (K), divides each by $\frac{1}{\sqrt{d_k}}$, and applies a softmax function to obtain the weights on the value vectors.

When the encoder is learning the embeddings for a word at a certain position, the attention score determines how much focus it should place on other parts of the input sentence.

This enables BERT to understand the meaning and context of each word based on its surrounding words.The position-wise feed-forward network applies a non-linear transformation to each position separately and identically. It consists of two linear layers with a ReLU activation function [Aga18] in between. This network allows BERT to model complex non-linear relationships between words in the input sequence.

Now, coming to the feature embedding in BERT, the model utilizes token embeddings, segment embeddings, and position embeddings(see Figure 4.5.). These embeddings help in representing the input text and incorporating the positional and contextual information.

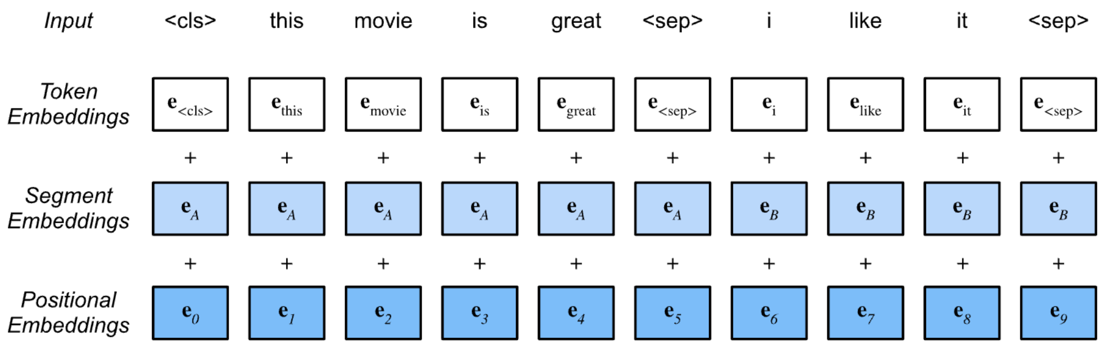

Input
<cls>
this
movie
is
great
<sep>
i
like
it
<sep>
Token Embeddings
$\mathbf{e}_{\text{<cls>}}$ $\mathbf{e}_{\text{this}}$ $\mathbf{e}_{\text{movie}}$ $\mathbf{e}_{\text{is}}$ $\mathbf{e}_{\text{great}}$ $\mathbf{e}_{\text{<sep>}}$ $\mathbf{e}_{\text{i}}$ $\mathbf{e}_{\text{like}}$ $\mathbf{e}_{\text{it}}$ $\mathbf{e}_{\text{<sep>}}$
Segment Embeddings
$\mathbf{e}_A$ $\mathbf{e}_A$ $\mathbf{e}_A$ $\mathbf{e}_A$ $\mathbf{e}_A$ $\mathbf{e}_A$ $\mathbf{e}_B$ $\mathbf{e}_B$ $\mathbf{e}_B$ $\mathbf{e}_B$
Positional Embeddings
$\mathbf{e}_0$ $\mathbf{e}_1$ $\mathbf{e}_2$ $\mathbf{e}_3$ $\mathbf{e}_4$ $\mathbf{e}_5$ $\mathbf{e}_6$ $\mathbf{e}_7$ $\mathbf{e}_8$ $\mathbf{e}_9$
+

Figure 4.5: The embeddings of the BERT input sequence are the sum of the token embeddings, segment embeddings, and positional embeddings.

Token embeddings: Each word or token in the input sequence is represented by a token embedding. These embeddings are learned during the model's training process and capture the semantic meaning of the tokens.

Segment embeddings: BERT can accept two input sequences, such as sentences or question-answer pairs. To distinguish between these sequences, segment embeddings are used. They assign different embeddings to the tokens of each sequence, allowing the model to differentiate and understand the relationships between different segments.

Position embeddings: Since the transformer architecture does not inherently capture the order of words in the input sequence, position embeddings are employed. They encode the relative or absolute position of each token in the sequence, providing the model with positional information.

By combining token embeddings, segment embeddings, and position embeddings, BERT can create rich, contextualized representations of words, considering both their meaning and their position in the input sequence. This enables the model to understand and generate representations that capture the complex semantics and dependencies within the text.

The use of feature embedding in BERT is very beneficial for us because it allows the model to overcome the limitations of traditional word embeddings that lack contextual information. By incorporating contextualized feature embeddings provided by BERT, we can assume that the feature embedding vector contains the information about the meaning of words based on their surrounding context.

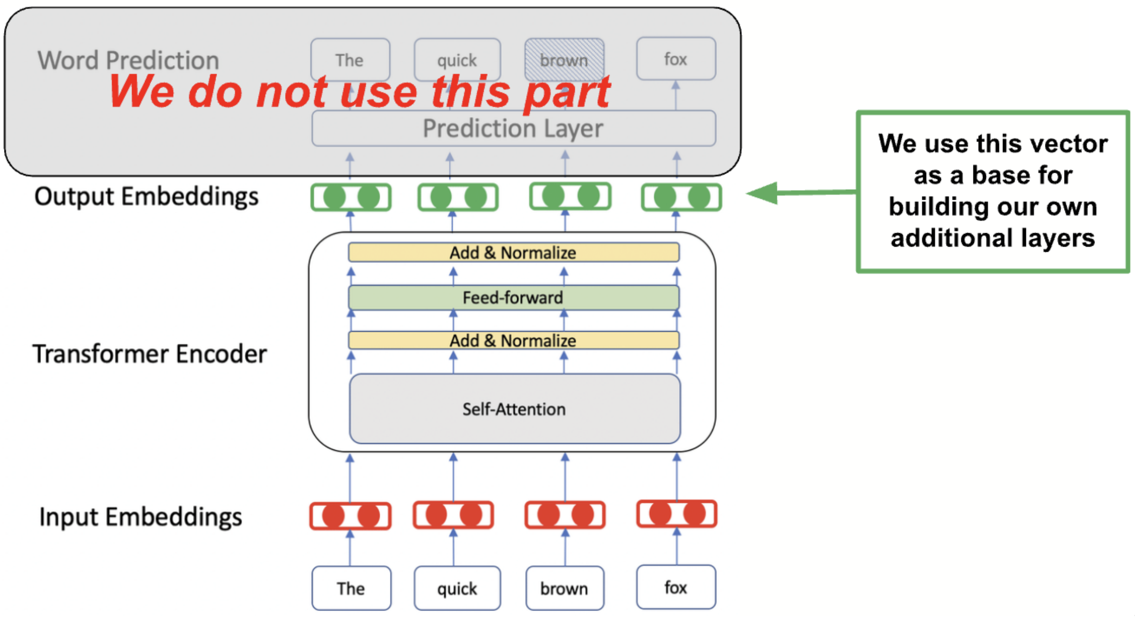


Figure 4.6: An illustration of the BERT model. The input text is first embedded into vectors (Input Embeddings) and then processed by the Transformer Encoder. The output is a set of extracted feature embeddings (Output Embeddings), which are then used as features for making a prediction or, in our case, for contrastive learning.

In the Figure 4.7 we provide a neural network architecture that takes an input tensor (which represents tokenized input text) and returns three outputs: logits, pooled output vector, and the last hidden state of the model. We call it MyModel further in the text.

The input tensor is fed into a pre-trained SciBERT model to obtain the last hidden state, which is a tensor containing the output features of the final layer of SciBERT for each token in the input sequence. The pooled output is then obtained by applying a max-pooling operation along the sequence dimension of the last hidden state tensor. The pooled output represents a fixed-size representation of the entire input sequence, which is then passed through three fully-connected layers (fc1, fc2, and fc3) to obtain the final output logits.

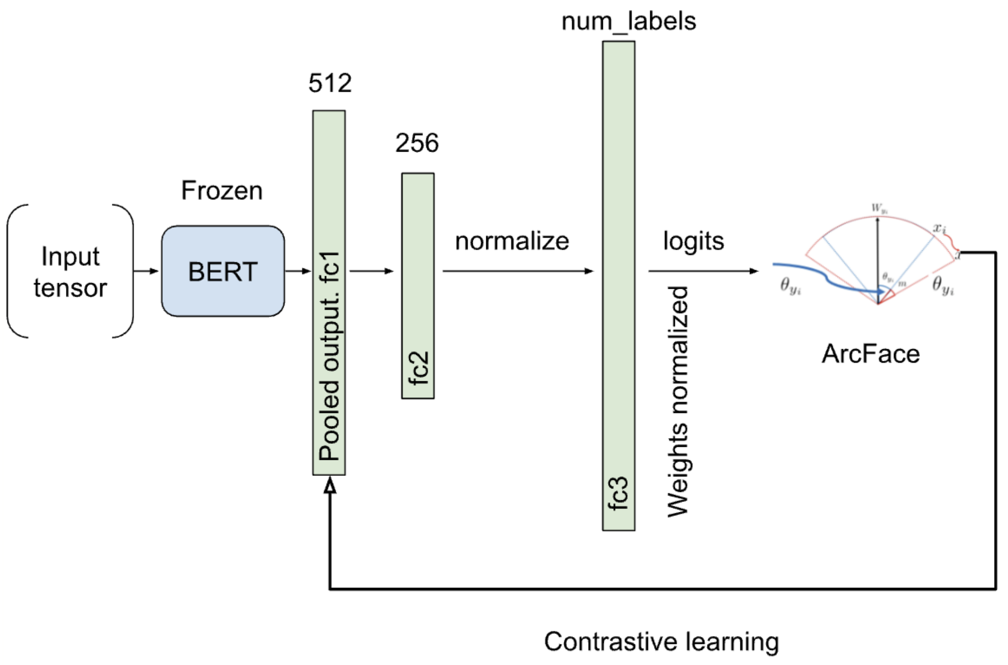


Figure 4.7: The architecture of MyModel with ArcFace for contrastive learning.

ArcFace loss function aims to improve the discriminative power of the learned embeddings by encouraging the cosine similarity between embeddings of the same class to be higher than the cosine similarity between embeddings of different classes. This is achieved by computing a margin-based softmax loss on the cosine similarity between the embedding and the weight vector of each class, where the margin is defined as the difference between the cosine similarity of the embedding and the weight vector of the true class and a fixed margin value m.

In MyModel 4.7, the fc3 layer is used to compute the final logits. The weight vector of the fc3 layer is normalized to unit length, and the embeddings produced by the model are also normalized to unit length before being used as inputs to the ArcFace loss. This ensures that the cosine similarity between the embeddings and the weight vector of each class is in the range [-1, 1]. In this architecture MyModel and ArcFace loss function work together to learn discriminative embeddings for text classification tasks by maximizing the cosine similarity between embeddings of the same class and minimizing the cosine similarity between embeddings of different classes.

It is important to mention that we are not training the SciBERT itself, we use it to generate the word embedding only.

4.6 Validating Contrastive Loss on MNIST and Its Potential for Complex Datasets

By testing our contrastive loss function on the MNIST dataset [LC05], we were able to validate its performance on a well-established benchmark and demonstrate its potential for use on other datasets. This is important because training models on our real dataset of abstracts is computationally and time-consuming, and testing on a smaller, standardized dataset like MNIST can help us identify potential issues and optimize our model before deploying them on larger, more complex real dataset.

The MNIST dataset contains images of handwritten digits from 0 to 9, and the goal is to train a model that can classify these digits correctly. It is commonly used as a benchmark for testing and comparing the performance of machine learning algorithms. There are several reasons why MNIST is a popular choice for testing neural networks:

- Simplicity: The MNIST dataset is relatively simple and easy to work with. Each image is a grayscale 28x28 pixel image of a handwritten digit, making it easy to load and preprocess.
- Size: The MNIST dataset is small compared to many other image datasets, with only 60,000 training examples and 10,000 test examples. This makes it easier to train and test models on a standard laptop or desktop computer.
- Familiarity: The MNIST dataset has been used for many years and has become a standard in the field of machine learning.

Difficulty: Although the MNIST dataset is relatively simple, it is still a challenging problem for machine learning algorithms. Achieving high accuracy on the MNIST dataset requires the development of effective feature extraction and classification techniques, making it a good test bed for evaluating the performance of different algorithms.

One additional reason why MNIST is commonly used for testing and comparing the performance of neural networks is its standardized data format. The images are already preprocessed, labeled, and split into training and test sets, which eliminates the need for additional data preprocessing and labeling.

This allowed us to focus on developing and testing the model and our loss function without worrying about data formatting. Furthermore, since the MNIST dataset is widely used and publicly available, using MNIST for testing is the ability to reproduce and compare results across different research studies. Since the dataset is standardized, other researchers can easily compare the performance of their models with others in the field. This can lead to more rigorous evaluation of results and facilitate the development of new techniques for improving performance.

In order to train a neural network model on the MNIST dataset we used PyTorch and PyTorch Lightning. PyTorch Lightning is used to train the model, which provides a higher-level API for training PyTorch models and takes care of many of the details of the training loop.

The model used is a neural network with three linear layers, where the first two layers have ReLU activation functions. The output of the second layer is passed through a normalization step before being fed into the third layer.

We used our custom implementation of the ArcFace loss function, and the Adam optimizer. The code is available on Github `https://github.com/iSenya/scibert-reference-recommendation`.

The training loop includes both training and validation steps, and logs the training loss, training accuracy, validation loss, and validation accuracy to TensorBoard logs.

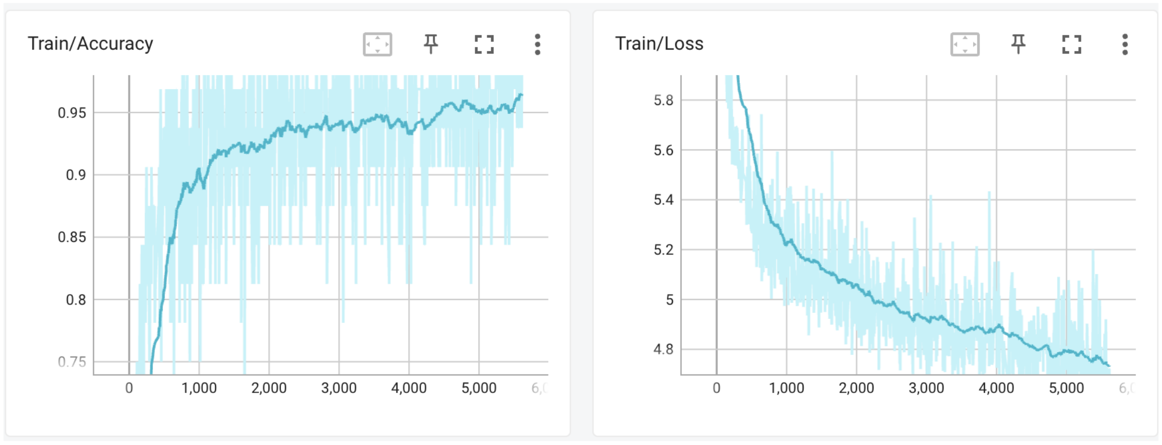


Figure 4.8: The training accuracy on MNIST with ArcFace loss shown on TensorBoard.

In 3 epochs we achieved great results on accuracy while the loss was still falling (Figure 4.8). The validation accuracy was still increasing, and the validation loss was falling down. Potentially, we could increase the number of epochs, and continue training. But our goal was to find out did we code the loss function correctly or not. Judging by results, it seems that we did not make any mistakes in the formula.

It's also important to note that while achieving good accuracy on MNIST is a great accomplishment, it may not be a sufficient benchmark for more complex datasets. As the complexity of the dataset increases, the performance of the model may decrease, and additional techniques such as regularization, data augmentation, or changing the architecture of the neural network may be needed to improve the performance. The successful imple-

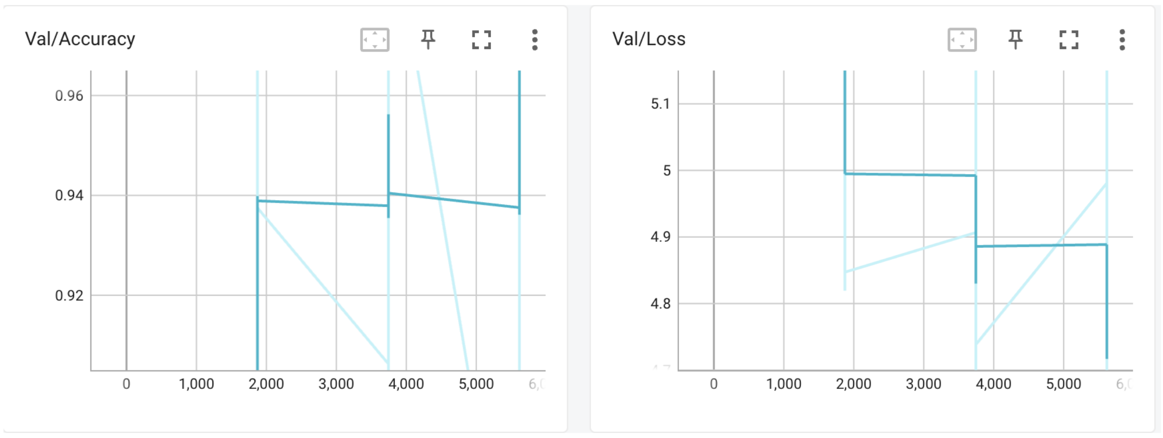


Figure 4.9: The training accuracy on MNIST with ArcFace loss shown on TensorBoard.

mentation of a custom loss function on MNIST is a promising first step towards applying it to more complex datasets, but further experimentation and evaluation are necessary to determine its effectiveness in our domain.

4.7 Creating Citing-Referencing Hyperspace Dataset with Semantic Scholar API

To create our citing-referencing hyperspace dataset for research papers, we needed to find the best source for gathering the required information and decide on the dataset's structure. While there are several databases available, such as Google Scholar, PubMed, and arXiv, we opted to use the Semantic Scholar API for our data retrieval.

Semantic Scholar stood out for providing a wealth of metadata about research papers, including abstracts, authors, publication venues, and references. It even offered features like citation counts, which helped us assess the impact of each paper. By employing natural language processing and machine learning techniques, Semantic Scholar enhanced the accuracy and relevance of our search results, making it a valuable resource for our research.

However, it's important to acknowledge the limitations of the Semantic Scholar database. One drawback is that it may not cover all scientific disciplines comprehensively, which means some fields or subfields might have limited representation in our dataset. There's also a bias in the selection of publications, with certain journals or publishers being favored over others. Additionally, Semantic Scholar primarily focuses on English-language articles, which could introduce a bias if we plan to extend our research to other languages.

Considering these limitations, we should complement our work by using additional sources and databases to ensure a more balanced and comprehensive understanding of the scientific literature.

After choosing the Semantic Scholar API, we utilized the Insomnia tool to retrieve data for specific paper IDs. This allowed us to gather various details for each paper, including its title, abstract, authors, publication venue, and references. This information formed the core of our dataset, providing a solid basis for exploring the citing-referencing hyperspace.

In order to create the database of abstracts, we started with the paper "Attention is all you need" [VSP+17] as our initial paper for requests to the API. We chose this paper because of its significant impact on the field of NLP, being cited more than 74 000 times. As we retrieved information about the paper, including its citation count and the papers that cite it and that it cites, we used this data to inform our research and provide suggestions for additional citations.

To create the dataset for learning, we went through the references and citations of the 1038 papers that we retrieved from "Attention is all you need". These papers became our "source papers", and we considered their abstracts as starting points for learning. We then iterated over each source paper and added the abstracts of its references to learn the dependencies between the abstract of the source paper and the abstracts of its references.

We made a conscious decision to stick to using abstracts instead of full text mining for two reasons: to avoid the time-consuming and computationally expensive process of web scraping and to reduce the amount of data that we needed to process, as working with abstracts alone would still provide us with sufficient information for our research purposes.

By using the Semantic Scholar API and the paper "Attention is all you need" as our starting point, we were able to create a dataset of abstracts for learning and inform our research with additional data and suggested citations.

The code then constructs a query to retrieve the data for the specific paper ID and sends a GET request to the API. If the request is successful, it extracts the paper's ID, title, abstract, and referenced abstracts from the response JSON and saves them in a list of extracted data. It also retrieves citations for the paper, and if there are more than 1000 citations, it retrieves them in batches using cursor pagination.

The code then constructs a list of paper IDs for the references and citations and uses a thread pool executor to asynchronously retrieve the paper data for each of these papers. The resulting paper data is combined with the extracted data from the original paper and printed as a JSON string. The code for it is available on Github as well.

`https://github.com/iSenya/scibert-reference-recommendation`.

The resulting structure of our dataset:

"paperId": "...",

"abstract": "...",

"referenced_abstracts": ["...", "...", "...", "...", ...]

After training the model on the dataset or citations and references of “Attention Is All You Need”, we decided to add to the dataset with the same structure other papers, the datasets can be accessed via google drive. These papers are about global warming - “Thermal Equilibrium of the Atmosphere with a Given Distribution of Relative Humidity” [MW67], and about biomedical research - “limma powers differential expression analyses for RNA-sequencing and microarray studies” [RPW+15]. Also, the last one had a lot of similarities in terms of vocabulary with the papers from computer studies, having such words as “expression”, “array”, “software”, “statistical”, “computing”, “data”, “analysis” in its abstract. We added this paper and its references to the dataset for several reasons. First of all, we assume that papers around “Attention Is All You Need” and around “limma powers differential expression analyses for RNA-sequencing and microarray studies” may cite each other. Also, their embedding vectors may be similar to each other due to the close vocabulary. However, they represent different areas of study. This should be visible in our research.

The full dataset we gathered consisted of 3136 papers. It is a relatively small dataset. And the topic distribution is around NLP, Climate Change, and Bioinformatics. We added the Climate Change topic consciously in order to add some noise to the data and let the model differentiate the clusters.

4.8 Enhancing Dataset Quality: A Preprocessing Pipeline

To improve the quality of our dataset, which serves as training data for a neural network model designed to generate abstracts for scientific papers, we developed a data preprocessing pipeline. Initially, the generated dataset suffered from various data quality issues, including empty abstracts, "None" values, and abstracts with unrecognized encoding or languages.

To improve the quality of our dataset, which serves as training data for a neural network model designed to generate abstracts for scientific papers, we developed a data preprocessing pipeline. Initially, the generated dataset suffered from various data quality issues, including empty abstracts, "None" values, and abstracts with unrecognized encoding or languages.

To address these issues, we implemented a series of text preprocessing steps in our pipeline. Firstly, we read in a collection of JSON files containing paper metadata and extracted the abstracts and references of each paper. The abstracts were then subjected to text cleaning functions that removed entities and hyperlinks. Additionally, we filtered the resulting set of abstracts to include only those written in English.

Next, we tokenized the preprocessed abstracts using the SciBERT tokenizer. This process converts the abstracts into a sequence of integer IDs that represent the input text. These tokenized abstracts were then paired with their respective paper IDs, forming the input data for training the neural network model.

To optimize the training process further, we divided the training dataset into 32-item batches using the PyTorch DataLoader. This allowed the model to process multiple examples simultaneously, resulting in more efficient training.

Ensuring the representativeness and diversity of the dataset was crucial. However, due to computational limitations, we had to reduce the initial dataset and perform random sampling to create a dataset of pairs consisting of "paper_id: [tokenized_abstract]". This randomization helped prevent the model from overfitting to a specific subset of the data and improved its ability to generalize to new examples.

To provide an overview of the downstream process, we included a schematic representation of the pipeline at Figure 4.10.

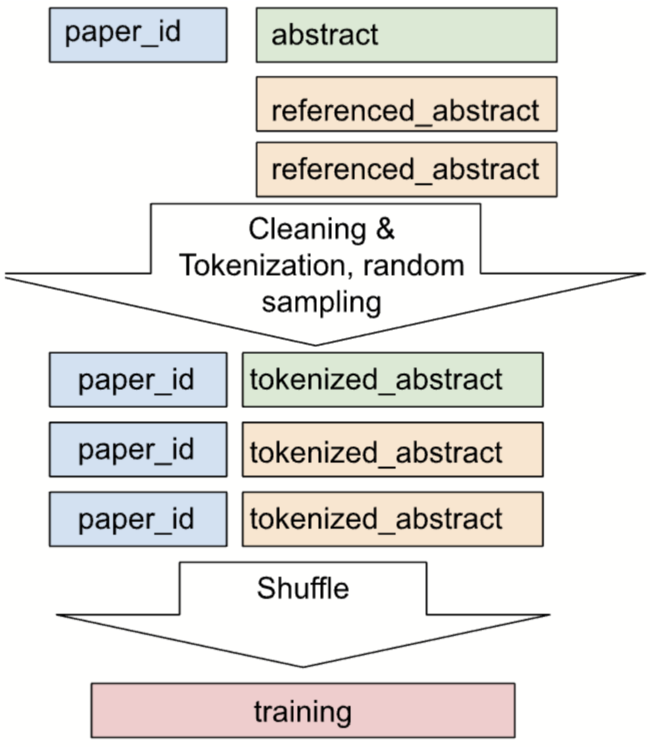


Figure 4.10: Data preprocessing pipeline for the training process.

4.9 Model Evaluation: Top-k Accuracy for Citation Suggestions

The accuracy function is used to evaluate the performance of a classification model. According to the way we defined the task, using the standard accuracy as a metric for model evaluation does not seem to be the right choice. Our task for the algorithm is formulated in a way that it should suggest a list of papers to cite or read. Ranging them in descending order.

In order to achieve that, we considered using the top-k accuracy. By considering the top-k predictions, the accuracy metric provides a more flexible measure of the model's performance. In addition, by allowing the user to specify the value of k, the accuracy metric can be customized to suit the specific needs of the task at hand.

We wrote the function for the top-k papers, where k is the optional parameter, depending on how we would fine tune the model in the future. The function takes as input the predicted logits (a tensor of shape [batch_size, num_classes]), true labels (a tensor of shape [batch_size]), and an optional parameter for the number of top predictions to consider (top_k).

The function computes the top-k predictions from the predicted logits, where k is the value passed in the top_k parameter. It then compares these top-k predictions with the true labels to determine the number of correct predictions. Finally, the function calculates the top-k accuracy of the model by dividing the number of correct predictions by the total number of examples in the batch.

4.10 Training the Model on a Small Sample

After cleaning our random samples we got 377 papers with abstracts, meaning 377 labels, out of which we got 16 430 pairs. During the training process, the model is fed with input data and it produces a set of output predictions. These predictions are then compared to the true labels of the data to calculate the difference between the predicted output and the true output. This difference is quantified using an ArcFace loss function which calculates the error of the predictions. The goal of training is to minimize this error and make the predictions as accurate as possible.

To minimize the error, the model parameters are adjusted through an Adam optimization algorithm to find the optimal values that result in the minimum error. This process is repeated iteratively over 20 epochs until the model's performance on the validation set no longer improves or the maximum number of epochs is reached.

During each epoch, the model is fed with mini-batches of size 32 instead of the entire dataset. This is done to reduce the computational complexity and memory requirements of the training process. After each mini-batch, the optimizer updates the model parameters using backpropagation, which calculates the gradient of the loss function with respect to the model parameters. The gradient indicates the direction of steepest descent towards the minimum error and the optimizer adjusts the model parameters in that direction.

Throughout the training process we monitor top-10 accuracy, and the training and validation loss to evaluate the performance of the model (Picture 7 and Picture 8).

We also used the scheduling for the learning rate (lr=3e-4) so that it reduces if the gradient reaches the plateau.

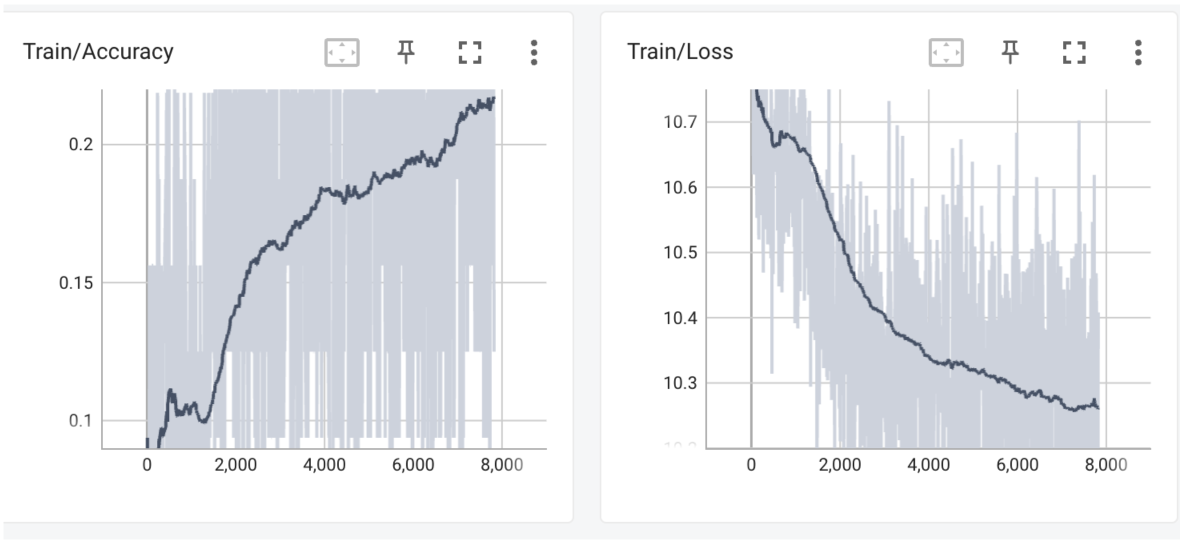


Figure 4.11: The training accuracy on real data with ArcFace loss shown on TensorBoard.

After 20 epochs we reached 22% of accuracy which is a good starting point for building the recommendation system on top of it, taking into consideration the small dataset we used, and our computational limitations.

4.11 Evaluating Citation Suggestions: Comparative Analysis

The accuracy metric alone is insufficient to evaluate the validity of suggestions provided by our model. To ensure the quality of the suggestions, we need to explore additional validation methods. One approach is to select an article that the model hasn't seen before and assess the model's ability to generate relevant citations in comparison to existing ones.

In order to proceed with this validation process, we will follow these steps:

1. Choose an article that our model hasn't encountered.

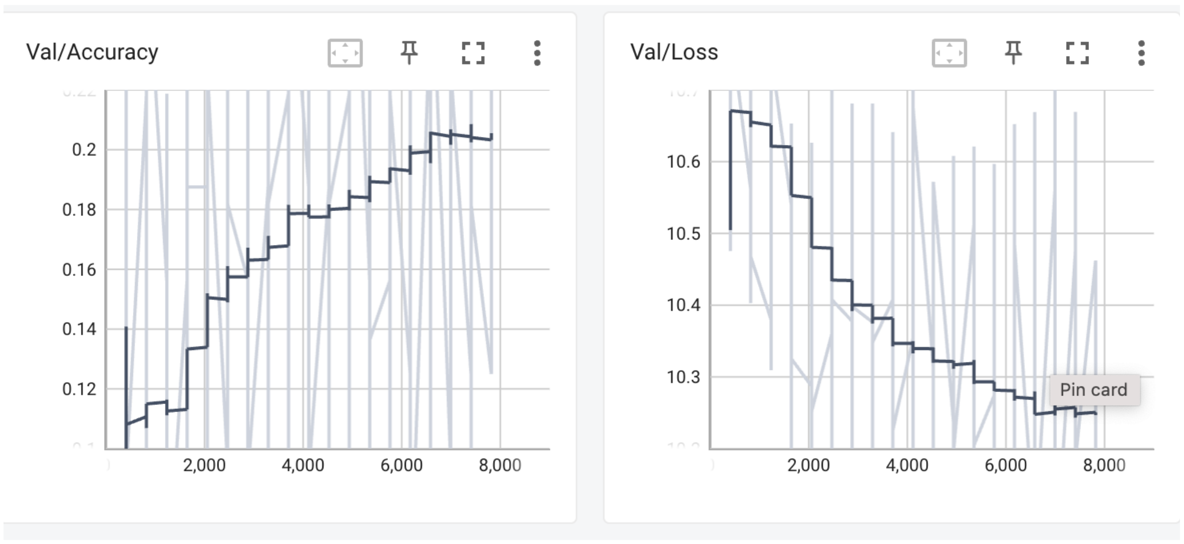


Figure 4.12: The validation accuracy on real data with ArcFace loss shown on TensorBoard.

2. Extract the abstract of the selected article.
3. Check if the article's references are available in the database.
4. Extract the references and their corresponding abstracts (if available) from the database.
5. Randomly select other articles from the database, including their abstracts.
6. Combine the references of the selected article with the randomly selected ones and shuffle the order.
7. Load the trained model checkpoint, which was trained using our customized loss function (MyModel).
8. Create embedding vectors with labels for the paper IDs and their titles.
9. Establish a base of embedding vectors for comparison.
10. Generate the embedding vector for the abstract of the chosen paper.
11. Compare the given vector with the vectors in the embedding base one by one using cosine similarity.
12. Sort the suggestions in descending order based on the cosine similarity scores, with higher scores indicating better suggestions.
13. Rate the suggestions using expert knowledge and human evaluation.

Considering that SciBERT was trained on specific domains, we have decided to select a paper from the Computer Science domain, potentially related to biomedical studies. The paper titled "Does Synthetic Data Generation of LLMs Help Clinical Text Mining?" [THJH23] meets these criteria. Additionally, it is important to ensure that the required information about its abstract, references, and abstracts of its references is present in the Semantic Scholar database for comprehensive evaluation.

By comparing the model's suggested citations against existing references we can determine the accuracy and relevance of the model's suggestions. A closer examination of the suggestions will shed light on their quality and effectiveness in providing valuable citations for further research.

The rating of the suggestions will be conducted through human expert knowledge, ensuring that the evaluation incorporates domain expertise and subjectivity that cannot be solely captured by automated metrics. By combining the strengths of both automated assessment and human evaluation, we aim to achieve a more robust and reliable validation process.

4.11.1 Creation of the Embedding Vector Dataset

Here are the next steps how we can generate reference suggestions based on the abstract we provide:

1. Convert the abstract into a format that our model can understand using our preprocessing pipeline. This involves converting the abstract into JSON format, cleaning, and tokenizing it.

2. Pass the abstract through our trained model and generate predictions. The model should output a list of suggested references that it believes are relevant to the abstract.

3. Evaluate the quality of the suggested references. We can do this by comparing the suggested references with known references for the abstract (if available) or by manually checking the relevance and quality of the suggested references as we already described.

There are several ways to further improve the quality of the recommendations. One way is to increase the size of the corpus by adding more papers. By doing so, the model can learn more about the relationships between papers and make more accurate recommendations.

Another way to improve the recommendations is to use the user's bibtex files. By incorporating the user's research interests, the model can make more personalized recommendations.

Using sections of the paper beyond the abstract, such as the introduction, conclusion, or discussion, may also lead to better recommendations. These sections contain more information about the paper's content and context.

Section labels can also be used as input to the model. By doing so, the model can learn to associate specific sections of the paper with specific topics, and make more precise recommendations.

It's worth noting that this approach may require multiple iterations to achieve the desired level of accuracy and relevance in the suggested references.

To implement this approach, we can start by loading a pre-trained model from a checkpoint file. Next, we remove the classification head from the model so that we can use it to generate embeddings. We then tokenize the desired section of the paper and generate the embeddings using the model. Finally, we can use these embeddings to find similar papers.

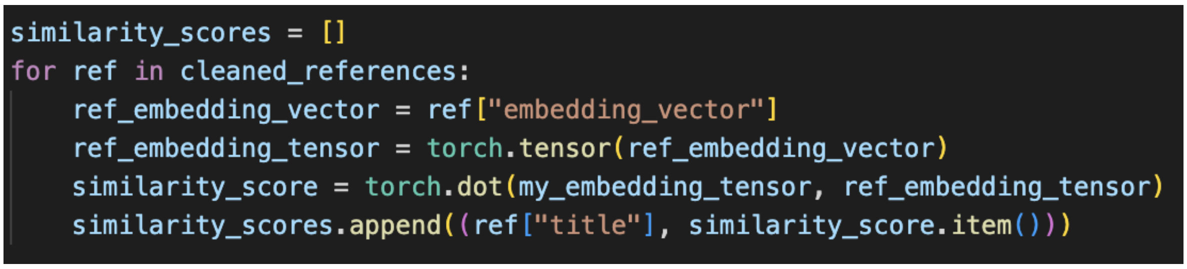
4.11.2 Evaluation Results

The algorithm passed through the references of the article "Does Synthetic Data Generation of LLMs Help Clinical Text Mining?", and cleaned them. After cleaning we got 33 true references and their embedding vectors. We repeated this procedure for several random papers, and added them to the dataset of embedding vectors. The total dataset contained 3910 papers and their embedding vectors.

The portion of true references in the evaluation dataset is 0.844%, which is a significantly low amount. The next step would be to check how the true references embedding vectors are distributed in the ranked recommendations. The ranked recommendations are computed with the help of the cosine similarity score. Taking into consideration that our vectors are already normalized, the similarity score is just the dot product of the two vectors - the initial one, and any other from the embedding vectors dataset. The code of computing the similarity score is as follows:

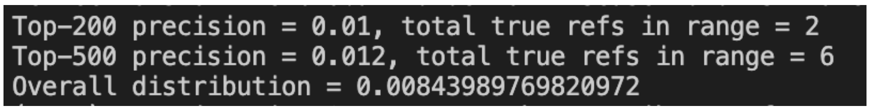
After computing the similarity scores, we performed ranging in descending order in order to find out did our recommendation better than random? How many true references got into the first top-500, for example. It turned out that the distribution among the top-500 is higher than the average. The portion of true references got into top-500 recommendations is 1.2% which is higher than the average 0.844% for our dataset. See Figure 4.14.

This evaluation says that our result is better than random, but is not enough to make a conclusion that our contrastive learning improved SciBERT. When we tried to do the same



```python
similarity_scores = []
for ref in cleaned_references:
    ref_embedding_vector = ref["embedding_vector"]
    ref_embedding_tensor = torch.tensor(ref_embedding_vector)
    similarity_score = torch.dot(my_embedding_tensor, ref_embedding_tensor)
    similarity_scores.append((ref["title"], similarity_score.item()))
```

Figure 4.13: Embedding vectors similarity score computation.



```
Top-200 precision = 0.01, total true refs in range = 2
Top-500 precision = 0.012, total true refs in range = 6
Overall distribution = 0.008439897698209 72
```

Figure 4.14: Distribution of recommendations in top-500 vs. distribution of recommendations in overall distribution.

exercise with the larger dataset, the results of the recommendation were also better than random. Top-500 precision was 0.8% versus overall dataset average 0.49% with dataset size of 6771 items. But when we plotted the true reference the shift of true references to the top recommendations was not noticeable with the human eye, even if we normalized the similarity scores between 0 and 1.See Figure 4.15.

Of course, we trained our contrastive loss only with three additional layers, and on an extra small dataset, if the model learned to shift related texts closer to the top recommendations, this shift is small. But this could be enough for us to understand if this game is worth playing further. Also, we can say that our proof of concept is achieved and it is worth training the model on a larger dataset and with more layers and parameters, or even retrain the whole SciBERT with another loss function (ArcFace), only if we check if our small transfer learning with contrastive loss improved the results of a pure SciBERT, trained with standard cross entropy.

First of all, we need to evaluate the results not only on one paper, but on a batch of papers, and average the score. In order to do this, we introduced the metric, which reflects how close are the referenced papers on average to the top of a recommendation list (best recommendations) for a given paper. For a single reference to a given paper the quality metric is calculated using the following formula:

$$Q = 1 - \frac{a}{b}, \tag{4.6}$$

where

a - is the distance between the true reference to the best recommendation from the

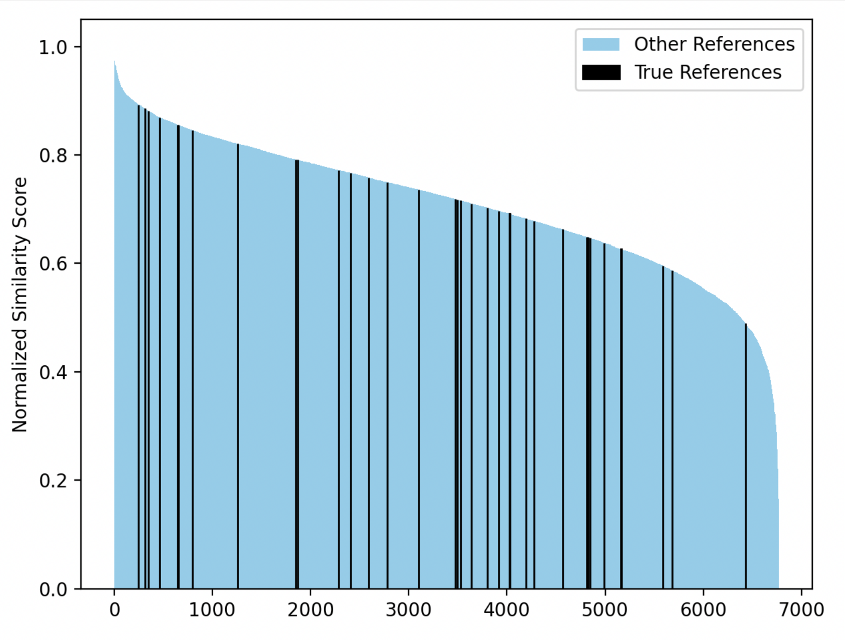


Figure 4.15: The distribution of the true references among other references and their normalized similarity scores to the initial paper

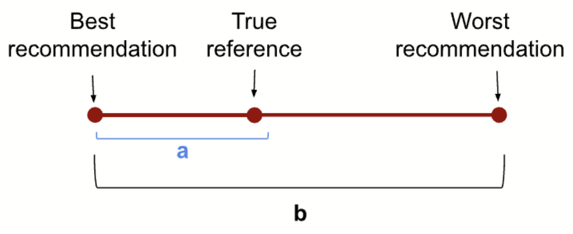


Figure 4.16: Abstract representation of the list of recommendations and the true reference in it.

sorted list in terms of position (see Figure 4.16),

b is the number of items in the recommendation list, so to say, the length of this list.

The score is then averaged over all references of a given paper and then averaged over all the papers, for which we are producing recommendations:

$$Q = \text{average}(q) \tag{4.7}$$

When all the references end up in the top of the recommendation list the quality metric will be close to 1. Conversely, when all the referenced papers end up close to the bottom of the recommendation list the metric will be close to 0. The value of 0.5 in this case would describe a model that gives random recommendations.

We calculated the quality metric for recommendations produced using the raw SciBERT model and using MyModel. The SciBERT model was found to produce recommendations described with the quality metric Q=0.55. The MyModel has the quality metric Q=0.63. This result demonstrated that the quality of recommendations is improved even when using a small dataset for fine-tuning and frozen weights of SciBERT. Drastically increasing the dataset size and unfreezing the weights of the underlying SciBERT model therefore has potential for further improvement of the recommendation quality.

Chapter 5

Discussion

SciBERT offers a powerful language model tailored for scientific text, with a focus on computer science and biomedical domains. While acknowledging potential biases in the training data, leveraging SciBERT's pretrained representations can greatly benefit the processing and analysis of scientific literature.

To enhance the fairness and inclusivity of SciBERT, efforts should be made to expand the training corpus to include a more diverse range of scientific disciplines. This ensures proportional representation of different subject areas, minimizing the bias introduced by imbalanced topic distributions. Additionally, incorporating strategies to mitigate publication and research biases can help foster a more equitable language model.

The addition of three fully connected layers on top of SciBERT, even with a very small dataset, improves the quality of recommendations. Therefore, it is necessary to unfreeze SciBERT weights and allow fine-tuning while substantially increasing the training dataset size.

One limitation of SciBERT (and BERT) is the input size constraint of 512 tokens. Abstracts, which are often longer, are truncated, resulting in information loss. Overcoming this limitation can be approached in several ways, such as exploring different architectures or processing the abstract sentence by sentence and then averaging or feeding them to another transformer.

It is important to note that references are not necessarily indicative of good recommendations. Some basic papers may serve as suitable references but are not expected to appear at the top of the recommendation list. Therefore, the quality of recommendations (represented by Q) is not expected to be 1, and determining a specific value of Q that is considered "good enough" can be challenging. However, tracking the performance of the models using Q can still provide valuable insights, as higher values indicate better recommendations.

General papers, often cited and widely known, are expected to be pulled towards different clusters by the contrastive loss algorithm. Consequently, they are not anticipated to appear at the top of the recommendation list. To observe this effect, it is imperative to have a large and well-connected training dataset in terms of references and citations. Thus, narrowing the domain for training should improve the quality of suggestions.

It is worth noting that abstracts may not contain all the necessary information to recommend relevant references. Considering more contextualized information for training, such as the paragraph preceding the appearance of references in a paper, could potentially yield better results.

The histogram of normalized similarity scores shows that the similarity between abstract texts is relatively high when converting them into embedding vectors. This implies that there is a need to focus on training dissimilarities and fine-tune the parameters of the loss function and training hyperparameters accordingly.

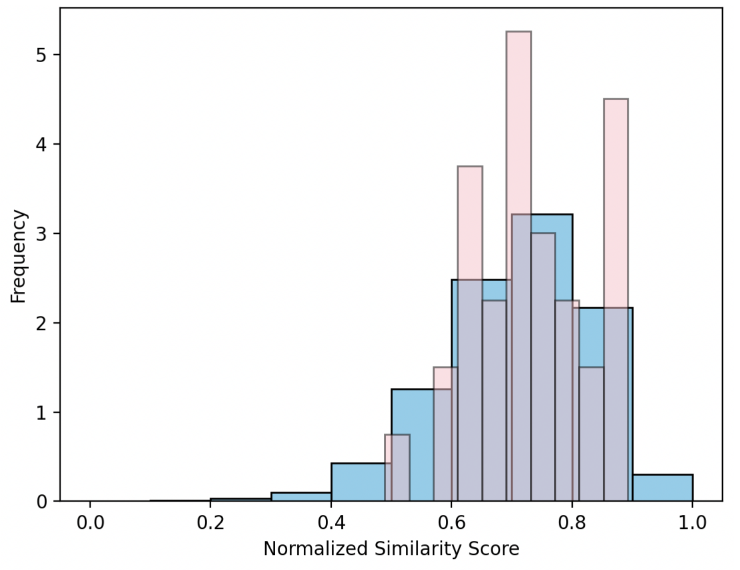


Figure 5.1: Histogram of Normalized Similarity Scores of Abstracts of True References and Random References to the Initial Abstract.

Figure 5.1 illustrates the histogram of normalized similarity scores for abstracts of true references and random references to the initial abstract. The distribution of scores between these two sets is nearly equal, suggesting that the model perceives abstract texts as highly similar based on their embedding vectors. Consequently, it is necessary to prioritize training on dissimilarities and adjust the parameters of the loss function as well as the training hyperparameters.

Chapter 6

Conclusion

In conclusion, this project focuses on harnessing transformers and attention mechanisms in Natural Language Processing (NLP) to enhance the generation of citation suggestions in the scientific research domain. Through an exploration of these techniques and an in-depth analysis of the SciBERT architecture, we have established the relevance and potential impact of transformers in NLP tasks.

The downstream process of this project encompasses several crucial steps. Firstly, we acquire a dataset of scientific papers and their associated citations, allowing us to extract the dependencies within the citations using open-source sources. This step provides us with the necessary information to train and fine-tune the SciBERT model for the citation suggestion task.

Applying transfer learning techniques, specifically with the utilization of contrastive loss, has proven successful in fine-tuning SciBERT for citation suggestion. Experimental evaluations have shown promising results, indicating that the generated citation suggestions exhibit a slightly higher density of true references among the top recommendations compared to the overall average.

Looking ahead, retraining SciBERT with contrastive loss stands out as a critical next step. By unfreezing the model's weights and substantially increasing the size of the training dataset, we expect further improvements in the quality and relevance of the citation suggestions. This approach offers an opportunity to leverage domain-specific knowledge and refine the model specifically for the citation suggestion task.

In addition to retraining, it is crucial to address other challenges that arise in the generation of citation suggestions. Expanding the dataset to encompass a broader range of scientific disciplines ensures proportional representation and minimizes biases introduced by imbalanced topic distributions. Overcoming input size constraints for abstracts is also important, as truncating longer abstracts can result in information loss. Exploring different architectures or processing abstracts sentence by sentence and then averaging or feeding them to another transformer are potential approaches to overcome this limitation.

By following a systematic approach and prioritizing the retraining of SciBERT with contrastive loss, we aim to advance the field of generating citation suggestions in scientific research. The utilization of transformers and attention mechanisms in NLP presents a promising avenue for improving the processing and analysis of scientific literature. With continued research and development, including efforts to expand the training corpus, mitigate biases, and address input size constraints, we can empower researchers and practitioners with more accurate and relevant citation recommendations. By facilitating the advancement of scientific knowledge and collaboration, we contribute to the growth and innovation of various scientific disciplines.

6.1 Source code

`https://github.com/iSenya/scibert-reference-recommendation/`

Bibliography

[Aga18] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[BLC19] Iz Beltagy, Kyle Lo, and Arman Cohan. Scibert: A pretrained language model for scientific text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, March 2019.

[DCT19] Jacob Devlin, Ming-Wei Chang, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2019.

[DGZ19] Jiankang Deng, J. Guo, and S. Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019.

[GF13] W. H. Gomaa and A. Fahmy. A survey of text similarity approaches. *International Journal of Computer Applications*, 68(13):18–28, April 2013.

[GNW21] John Giorgi, O. Nitski, and Bo Wang. Declutr: Deep contrastive learning for unsupervised textual representations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 849–862. Association for Computational Linguistics (ACL), August 2021.

[GYC21] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6962–6974. Association for Computational Linguistics (ACL), November 2021.

[HA14] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on...* Springer, December 2014.

[JWF18] H. Jelodar, Yongli Wang, and Xia Feng. Latent dirichlet allocation (lda) and topic modeling: models, applications, a survey. *Multimedia Tools and Applications*, 77(23):30631–30665, 2018.

[JXKS22] Minghang Ju, Yanyan Xu, Dengfeng Ke, and Kaile Su. Masked multi-center angular margin loss for language recognition. *EURASIP Journal on Audio, Speech, and Music Processing*, 2022(17), 2022.

[Kon05] Grzegorz Kondrak. N-gram similarity and distance. In *String Processing and Information Retrieval*, pages 115–126, November 2005. Published in SPIRE.

[LC05] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits. *Pattern Recognition Letters*, 26(6):771–780, 2005.

[MW67] S. Manabe and R. Wetherald. Thermal equilibrium of the atmosphere with a given distribution of relative humidity. *Journal of the Atmospheric Sciences*, 24(3):241–259, May 1967.

[PRZS16] Christian Paul, Achim Rettinger, Jens Ziegler, and Pedro A. Szekely. Efficient graph-based document similarity. In *Extended Semantic Web Conference*, pages 313–328. Springer, May 2016.

[QS17] Ce Qi and Fei Su. Contrastive-center loss for deep neural networks. In *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017.

[Ram03] J. E. Ramos. Using tf-idf to determine word relevance in document queries. *Computer Science*, 2003.

[RG19] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3982–3992. Association for Computational Linguistics (ACL), August 2019.

[RPW$^+$15] M. Ritchie, B. Phipson, Diabetes-Ling Wu, Yifang Hu, C. Law, W. Shi, and G. Smyth. limma powers differential expression analyses for rna-sequencing and microarray studies. *Nucleic Acids Research*, 43(7):e47, January 2015.

[RS19] Colin Raffel and Noam M. et al. Shazeer. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

[THJH23] Ruixiang Tang, Xiaotian Han, Xiaoqian Jiang, and Xia Hu. Does synthetic data generation of llms help clinical text mining? *ArXiv preprint arXiv:2303.04360*, March 2023.

[VSP$^+$17] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

[WWe18] Hao Wang, Yitong Wang, and et al. Cosface: Large margin cosine loss for deep face recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.

[ZS18] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *32nd Conference on Neural Information Processing Systems (NeurIPS 2018)*, 2018.