LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN
CENTRUM FÜR INFORMATIONS- UND SPRACHVERARBEITUNG
STUDIENGANG COMPUTERLINGUISTIK

Masterarbeit

im Studiengang Computerlinguistik

an der Ludwig- Maximilians- Universität München

Fakultät für Sprach- und Literaturwissenschaften

Learning improved rare word representations in pre-trained language models using definitions

vorgelegt von
Marcel Braasch

Betreuer:	Lütfi Kerem Şenel
Prüfer:	Prof. Dr. Hinrich Schütze
Bearbeitungszeitraum:	16. September 2022 - 31. Januar 2023

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig angefertigt, alle Zitate als solche kenntlich gemacht sowie alle benutzten Quellen und Hilfsmittel angegeben habe.

München, den 30. Januar 2023

.....
Marcel Braasch

Danksagung

Diese Arbeit ist für meine Eltern, Emeli und Jens. Danke für alles, ich liebe euch!

Abstract

Deep pre-trained language models (PLM) have revolutionized natural language processing. Through the emergence of BERT, GPT-3 and XLNet - to name only a few - boundaries of what was thought would be possible in natural language understand and generation have been pushed. These models are able to solve a range of difficult tasks on par with human baselines, even exceeding them in many. Machine translation, text summarization, sentiment analysis or named entity recognition barely pose any problems for these types of models anymore.

They do, however, lack understanding of rare and domain specific words. PLMs are trained on massive amounts of textual data. Words which occur frequently, in various realizations, senses and contexts are simple to understand. Rare and domain specific words are not. While *deoxyribonucleic* is a common word in genetics it is rarely ever used in day-to-day language. In addition to that, PLMs typically have a limited vocabulary size. Out-of-vocabulary words are assembled by their subtokens such that the model can make sense of the holistic meaning by its subparts. This approach assumes that subword information is sufficient to infer the meaning of the entire word. Likely, this is a simplifying assumption. For example, BERT's tokenizer will split up *deoxyribonucleic* into *de*, *##ox*, *##yr*, *##ib*, *##on*, *##uc*, *##lei*. While tokens like *de-* may denote prefix information many of the previously shown do not or barely carry inherent information.

To overcome this downfall, this works assesses methods to learn better representations for words of this type. I train a BERT model to predict what would be the one-token-representation (OTRs) of a word which is as of now split into subunits by the tokenizer. This is done by solely showing the model a set of definitions scraped from various dictionaries and asking it to reconstruct its OTRs of know OTRs. Once trained, I infer the OTRs for rare words which would otherwise have been split into multiple subtokens. This approach I call Defbert.

Defbert is evaluated on a range of challenging tasks. In specific settings Defbert outperforms the baseline BERT model. Moreover, it is on par with other approaches addressing the rare-word-problem. Apart from that, I showcase Defbert's abilities in a few-shot and zero-shot setting. Unfortunately, I was not able to show an increase in performance on downstream tasks, however, point to a wide range of possibilities to continue assessing the idea.

Contents

Abstract I

1 Introduction 3
- 1.1 Motivation . . . 3
- 1.2 Aim . . . 4
- 1.3 Methods . . . 4
- 1.4 Overview . . . 4

2 Fundamentals 7
- 2.1 Attention . . . 7
- 2.2 Transformers . . . 7
- 2.3 Pre-Trained Language Models . . . 8
  - 2.3.1 BERT . . . 9
  - 2.3.2 Word Embeddings . . . 10
  - 2.3.3 Tokenizers . . . 11
- 2.4 Zero-Shot and Few-Shot Learning . . . 11
- 2.5 MMR . . . 12
- 2.6 Related Work . . . 13
  - 2.6.1 Fine-tuning on a dataset specific to the domain . . . 13
  - 2.6.2 Using domain-specific language models . . . 13
  - 2.6.3 Using domain-specific word embeddings . . . 14

3 Defbert 17
- 3.1 Approach . . . 17
- 3.2 Definitions . . . 17
- 3.3 Patterns . . . 21
- 3.4 Training . . . 22
  - 3.4.1 Ordinary Defbert Setting . . . 23
  - 3.4.2 Hierarchical Defbert . . . 23
- 3.5 Evaluation . . . 25
  - 3.5.1 Defbert on WNLaMPro . . . 25
  - 3.5.2 Defbert on Scientific Text Classification . . . 26
  - 3.5.3 Hierarchical Defbert on WNLaMPro . . . 27
  - 3.5.4 Defbert's Few-Shot Abilities: Binary Classification . . . 28
  - 3.5.5 Defbert's Zero-Shot Abilities: Exact Pattern Matching . . . 29
- 3.6 Discussion . . . 32

4 Conclusion 33
- 4.1 Summary . . . 33
- 4.2 Outlook . . . 33

Bibliography 35

1 Introduction

Deep pre-trained language models (PLM) equipped with Transformers currently dominate the NLP landscape [14, 9, 64]. They follow a clear paradigm. Models are pre-trained in a self-supervised fashion given an abundance of textual data [14]. Typically pre-training objectives are masked-language modelling or next-sentence-prediction [14], corruption detection [**?**] or word-order-prediction [26]. Next, these models are fine-tuned and evaluated on downstream benchmarks such as GLUE and SQuAD [46, 65]. A new paradigm which recently emerged is prompt-based fine-tuning [9]. Instead of training the model in classic gradient-based manner, the task to be solved is prompted directly with a set of input samples. Large (and some small) PLMs combined with correctly engineered prompts achieve comparable if not superior performance to previous approaches [49, 63].

While these approaches yield stunning results and are proof of excellent engineering efforts, they have a range of shortcomings which are being addressed by the scientific community. Widely known pitfalls are a lack of common-sense understanding, complex reasoning capabilities or a simple understanding of our physical world [45, 7].

Besides these, two other flaws are the lack of domain specific knowledge [15, 81], and insufficient understanding of rare words [71, 50]. In simple applications such as sentiment analysis for movie reviews [4], or named entity recognition for social media current machine learning models perform extraordinarily well [1]. If employed in more specific settings such as biology, finance or law there are large gaps to fill [61, 10].

1.1 Motivation

While pre-trained language models are capable of solving a wide range of tasks they usually do not perform well in situations of which they haven't seen the suitable data or simply their quantity was not sufficient [48]. The quality of the language model and its understanding of words is typically determined by the contexts of the word that have been seen during training. Similarly to other machine learning models, if new out-of-domain samples are encountered, the models fails to work well.

PLMs posses extraordinary linguistic knowledge. Despite being trained on millions of examples they lack understanding of rare and domain specific terminology. One main issue is that PLMs typically have a limited vocabulary size. Out-of-vocabulary words are therefore assembled by their subtokens. Many deep language models including BERT make use of byte-pair encoding [55], WordPiece [74], or similar subword tokenization algorithms [48]. This approach assumes that subword information is sufficient to infer the meaning of the entire word. Likely, this is a simplifying assumption, which will be further explained in the following section.

Due to the lack of the models' understanding of rare words they often times underperform in settings where domain knowledge is required [27, 76]. Specially looking at real-life applications and use-cases in industries domain-knowledge is indispensable. There are a range of techniques addressing specific enhancing models with specific industry or domain knowledge but these techniques are typically not very flexible in usage since they were specifically design for a single purpose [2, 48]. Thus, we are looking for a one-fits-all solution flexible to adapt to any situation imaginable increasing PLMs performance in challenge situations.

1.2 Aim

The aim of this thesis is to enhance PLMs with better representations of rare and domain specific words. Infrequent words are broken up into subtokens which are supposed to carry an inherent meaning. By the respective subtoken's representation and its conglomerate BERT is supposed to infer its meaning on the fly. For many words and its representations its true that their sub-tokenized embeddings yield meaningful representations. For example, regard the word *appreciations*. BERT's tokenizer will break the word into *appreciation* and its plural suffix *##s*. Many grammatical phenomena and frequently occurring events will be satisfactorily represented by this sub-tokenization. The model will have no issue understanding the meaning of the suffix *##s* in the vast majority of cases. This is mainly due to its frequent occurrence.

Often this fact does not hold. Regard a word like *Deoxyribonucleic*. In genetics, this word is not a rare word at all. However, in day-to-day speech it is not used very often. The BERT tokenizer will break the word into the subtokens *de*, *##ox*, *##yr*, *##ib*, *##on*, *##uc*, *##lei*, *##c*. Clearly, breaking up the word into eight chunks carrying relatively low meaning will not improve the representation of *Deoxyribonucleic*.

The aim of this thesis is to overcome this shortcoming. We developed an approach called Defbert. We plan to find a universal approach applicable to use cases where rare word or domain specific knowledge is required. This would impede having to develop domain specific language-model approaches [27, 2, 76], or may be breeding ground for research in domain specific natural language processing.

Besides, this thesis shall highlight previous work conducted to tackle this issue and can serve as a starting point to get an idea of the problem at hand. We present a broad overview of methods already thoroughly evaluated and build upon these Moreover, we showcase a range of techniques to evaluate PLMs capabilities to overcome the issue explained.

1.3 Methods

The tools used in this work are mainly implemented using Python 3. Word and sentence tokenizer's are taken from the NLTK software package. The Defbert implementation is backed by huggingface's transformers library. Model training and deployment is conducted using PyTorch and PyTorch Lightning. To evaluate Defbert we make use of the script provided in the WNLaMPro repository. To scrape definitions from Wiktionary we use the requests library as well as the Beautiful Soup software package. All of the code written as part of this thesis can be found on the CD appended to the back of the printed version of this thesis.

This work is based on preliminary results of my supervisor Lütfi Kerem Senel. The results of his experiments are presented in Section 3.5.1 and 3.5.2. The efforts conducted in this thesis is a strong research focus of the the Center for Information and Language Processing (CIS) and Prof. Schuetze's research group, thus many ideas and possible next steps are based on previous findings [50, 51, 52, 54].

1.4 Overview

In Section 2 the fundamentals used throughout this work are presented. Transformers, PLMs including the model in use, as well as other techniques employed in this work are introduced. The work starts off with deep learning essential focusing on attention and transformers (Section 2.1 and 2.2 and are topped of with explanation on the model in use and related concepts in Section 2.3. Zero- and few-shot learning is briefly introduced in Section 2.4. In Section 2.6, the thesis moves on the introducing previous works this work is mainly based on.

In Section 3 the main contributions of this work are discussed. I present the high-level overview of the approach in Section 3.1. The training data at hand, namely word definitions, is introduced in Section 3.2. Section 3.3 explains the patterns used to conduct training. Section 3.4 explains the training procedure in depth and explains the implementation of a novel hierarchical training procedure. In Section 3.5 the approach is evaluated by the means of a range of methods and finally discussed in Section 3.6.

The work is eventually concluded in Section 4. I briefly summarize the most important contributions of the thesis and discuss the outlook with possible future steps in Sections 4.1 and 4.2, respectively.

2 Fundamentals

To better understand the inner workings of the backbone model used in this work, BERT, I will briefly recap attention [64] (Section 2.1), transformers (Section 2.2), pre-trained language models (Section 2.3), zero- and few-shot learning (Section 2.4). Lastly, work related to this thesis is presented in Section 2.6.

2.1 Attention

The concept of attention has been extensively studied by psychologists and neuroscientists for many years, with a focus on how it impacts human stimulus processing [62]. The deep learning community has also embraced this idea, borrowing inspiration from nature, as with many technological advancements.

The application of attention in deep learning models was first explored in the field of computer vision [3, 37, 75, 62]. The natural language processing community then applied this concept to state-of-the-art encode-decoder architectures [23, 39, 35, 5].

Incorporating attention into encoder-decoder models has been shown to significantly improve performance. Traditional architectures struggled with decoding long-term dependencies because they only encoded long sequences into a single vector representation. Attention mechanisms, on the other hand, allow for the important parts of a sequence to be remembered during both encoding and decoding, regardless of their relative position.

Attention mechanisms have allowed researchers to gain a deeper understanding of the previously enigmatic inner workings of neural networks. In particular, attention enables one to see which parts of an encoded sequence are attended to during the decoding of a particular output sequence. For example, in neural machine translation tasks, attention has been used to significantly improve performance, especially for long sequences [5]. Attention can also be used to align corresponding words in the input and target sequences. Attention can be loosely understood as a memory extension for models, enabling them to remember (or *attend to*) important relationships of e.g., words at each step of the underlying process.

2.2 Transformers

The introduction of attention mechanisms was a significant development, but encoder-decoder architectures, which are inherently sequential and non-parallelizable, still presented challenges [25]. In 2017, the paper "Attention is all you need!" by Vaswani et al. [64] introduced a new paradigm called transformers, which aimed to address these issues. The following subsections will outline the main ideas of this paper and provide additional annotations on some of the concepts discussed.

Model Architecture

The transformer architecture has a key advantage over previous models in that it does not process sequences sequentially. Instead, the entire input sequence is encoded at once. The model consists of an encoder and a decoder, each of which comprises N stacked attention layers followed by simple feedforward neural networks. Residual connections are used to allow the integration of information at multiple levels of abstraction [16]. The output of each layer is concatenated with the unprocessed input and normalized. The

encoded input sequence is then passed to the decoder, which follows the same process as the encoder but uses masked multi-head attention in the first step. This disguises the right side of the sequence, which has not yet been processed, in order to maintain the model's autoregressive (left-to-right) behavior and prevent it from looking ahead. Finally, the decoded sequence is passed through a linear layer and softmax function to produce a probability distribution over the possible output sequence.

Scaled Dot-Product Attention

In essence, attention in the transformers model can be seen as a selection mechanism retrieving the most important information of an input sequence.

Matrices in the following should be seen as batched inputs of their respective subject. A query matrix Q is multiplied by a key matrix K. Vectors (the rows of Q and columns of K) which are close to each other result in a large value. This is followed by a scaling and softmax. This procedure yields a sparse matrix which can be seen as a selector to the value matrix V. It is important to note that Q, K and V are the result of the input matrix X and the respective weight matrices W^Q, W^K and W^V which are the subject of the training process (e.g., $W^Q X = Q$). More formally the scaled dot-product attention can be expressed as

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

where d denotes the dimension of the input embeddings.

In short, we use the Query Q to select the key K. This key K is used to select the appropriate value one should pay attention to.

Multi-Head Attention

In a simplified manner, multi-head attention can be understood as the concatenation over h scaled dot-product attention modules, referred to as *head*. Besides shared weights across all heads, each $head_i$ is initialized with their own weight matrices W_i^Q, W_i^K, W_i^V resulting in a mathematical expressing as

$$MultiHead(Q, K, V) = Concat(head_1, \ldots, head_h)W^O \tag{2.2}$$

$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.3}$$

$$= Attention(XW^QW_i^Q, XW^KW_i^K, XW^VW_i^V). \tag{2.4}$$

The transformers model include a few more important aspects. A more thorough explanation (e.g., discussing the positional encodings) is beyond the scope of this thesis and is therefore omitted. A graphic visualization of multi head attention, the transformers encoder unit and BERT can be found in Figure 2.1.

2.3 Pre-Trained Language Models

Pre-trained language models have significantly impacted the field of natural language processing (NLP) in recent years, leading to new state-of-the-art results on a number of benchmarks and practical applications in a variety of fields [9, 14]. These models are trained on large datasets and can then be fine-tuned for specific tasks or used as-is for tasks such as language translation, text classification, and question answering [46].

One of the key advantages of pre-trained language models is their ability to leverage large amounts of data to learn contextualized word representations that capture language structure and patterns. For example, models such as BERT and GPT-3 use exactly the attention mechanisms explained to effectively process long-range dependencies and

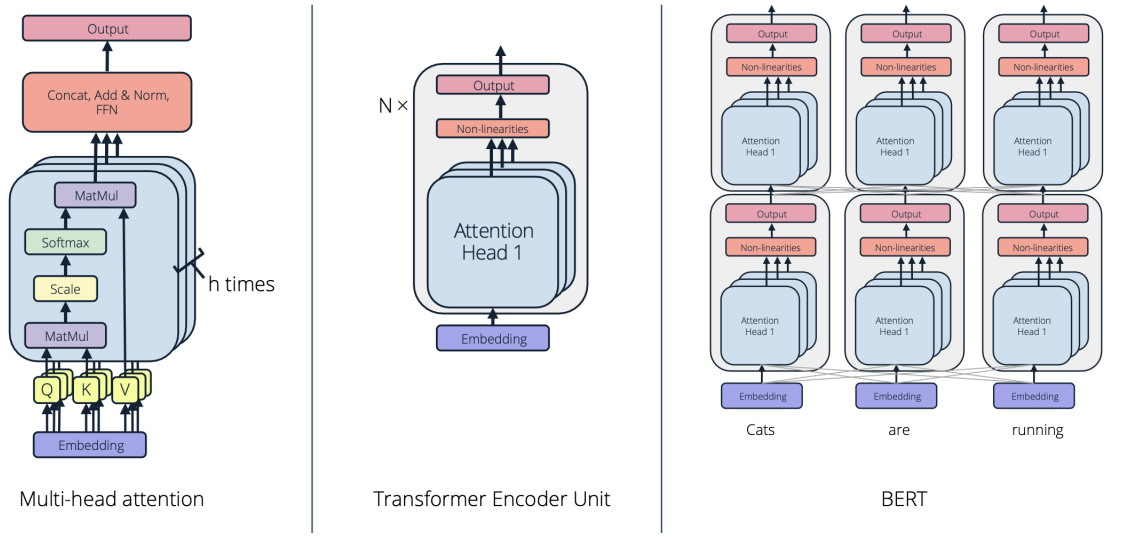

Figure 2.1: Visualization of multi-head attention (Section 2.1), the transformer encoder unit (Section 2.2) and BERT (Section 2.3.1).

contextual information [14, 9]. As a result, these models can achieve strong performance on a wide range of NLP tasks, even when fine-tuned on small datasets [46, 69].

In addition to their strong performance, pre-trained language models have the benefit of flexibility and transferability. They can be fine-tuned for a wide range of tasks and can be used to improve performance on tasks they were not specifically trained for. For instance, a model trained on a language translation task may also perform well on a text classification task due to the shared language structure and patterns that it has learned [14].

While pre-trained language models have proven to be highly effective, they also have some limitations. One of the main challenges is the requirement for a large amount of high-quality, task-specific data for fine-tuning. In addition, these models can be computationally expensive and may require specialized hardware for training and inference [20]. This specifically includes pre-training the models, as fine-tuning is comparably cheap. Another issue is the potential for bias in the data used to train the models, which can lead to biased outputs, if not addressed accordingly [6, 53].

Overall, pre-trained language models have greatly advanced the field of NLP and have opened up many new possibilities for practical applications. They offer a strong starting point for a wide range of NLP tasks, but it is important to consider their limitations and the potential for bias in the data used to train them.

2.3.1 BERT

Bidirectional Encoder Representations from Transformers, short BERT, is a pre-trained language model that has significantly impacted the field of natural language processing (NLP). BERT is trained on a large dataset of unannotated text and is designed to learn contextual relationships between words (or subwords) in a sentence. It was specifically designed to be a universal language model which is able to learn a wide range of tasks with minimal effort [14]. When BERT was first introduced, one of the key innovations were its use of a transformer architecture and self-attention mechanism [64], which allows it to effectively process long-range dependencies and contextual information. It can be fine-tuned for a variety of tasks with minimal task-specific architecture modifications, and has even been used to improve performance on tasks that it was not specifically trained for [14, 46].

Architecture

The model layout consists of two, very similar, steps. Step one, the pre-training, consists of feeding unlabeled tuples of subsequent sentences to the model. The objective BERT tries to solve during pre-training time is composed of two tasks. **1.** Sentences are masked according to a certain pattern and the respective masked word has to be predicted by the model. **2.** Given sentence A, BERT is asked to predict whether sentence B is actually following sentence A, or if it is a negative sample. Step two, the fine-tuning, initializes the model with the parameters from pre-training and, once again, gets fed tuples of sequences. This time, however, the tuples try to solve a very specific task. For instance, BERT could be fine-tuned to the NLP task of question answering. The input would then be a set of *(question, context, answer)* tuples. Recently, a second paradigm besides gradient-based fine-tuning has been discovered. Few-shot and zero-shot prompt-based fine-tuning seems like a promising research direction currently gaining a lot of attention [49, 9].

Previous models were only trained in an auto-regressive manner (either right-to-left or left-to-right), being able to only see one side of the context. BERT improves this with its fully connectedness enabled by the attention mechanism combined with the masked-language modelling objective by letting the model see the whole context the entire time. This, however, implies an important limitation. Since the model is informed about the whole context the entire time, *predicting* the masked word of a sequence would result in *knowing* the word to be predicted. No matter at which time step, the model would, in simple terms, *see itself*. To counteract this, during training time (and later during run time) a certain percentage of the tokens are masked. E.g., the sequence *'The cat likes to play'* could result in *'The cat [MASK1] to play'*. Besides this, there are a few other artificial tokens added to sequences, but a further explanation will be omitted as they are not too important to grasp the concept of the model.

The architecture of BERT is very simple and its name already reveals what lays behind it. It is comprised of a fully connected L-layer bidirectional transformer encoder neural network with A attention heads and a hidden size of H.

2.3.2 Word Embeddings

Word embeddings are a type of representation for NLP tasks that maps words or phrases from a vocabulary to a continuous vector space. These embeddings are learned from data and are designed to capture the semantic and syntactic relationships between words [36].

There are two main types of embeddings: contextualized and non-contextualized word embeddings. Non-contextualized embeddings, such as word2vec and GloVe [42, 36], represent words based on their overall distributional statistics within a dataset. In contrast, contextualized embeddings, such as BERT and ELMo [14, 41], represent words based on their context within a sentence or passage. This allows contextualized embeddings to capture a wider range of word relationships and nuances, making them more effective.

One of the main limitations of word embeddings is that they tend to have poor representations for rare and domain-specific words [48]. This is because these words may not occur frequently enough in the training data to effectively learn their meaning and relationships. From a technical perspective, this is also limited by the model's tokenizer.

Tokenizers in deep pre-trained language models often have limited vocabulary sizes because increasing the vocabulary size would require more memory and computational resources, which can make the model more expensive to run and slower to process text. One benefit of increasing the vocabulary size is that the model will be able to handle a larger number of unique words and potentially improve performance on tasks that involve rare words. However, this also means that the model will be more prone to overfitting on the training data and may perform worse on out-of-sample data. On the other hand, decreasing the vocabulary size can help reduce the size and complexity of the model, which can make it faster and cheaper to run. However, this may also result in the model

struggling to handle rare words and out-of-vocabulary words, as it will have fewer options for handling these words [38].

In general, the choice of vocabulary size is a trade-off between model performance and computational efficiency, and the optimal size will depend on the specific task and dataset being used. A more specific description of tokenizers and their use in the setting of this work is provided in the following section.

2.3.3 Tokenizers

Tokenizers are a fundamental component of pre-trained language models, as they are responsible for dividing text into individual tokens that can be processed by the system [74, 55]. Pre-trained language models, such as BERT (Bidirectional Encoder Representations from Transformers), rely on effective tokenization methods to accurately process a wide range of words and phrases [14].

One of the main challenges in tokenization is handling rare and out-of-vocabulary words, which may not be included in the model's fixed vocabulary. BERT addresses this issue through the use of a subword tokenization method called WordPiece [74]. WordPiece is a vocabulary-based tokenization that divides words into subwords and adds a special "piece" symbol to denote the end of a subword.

The WordPiece vocabulary is learned from the data during the training process and is designed to balance the need for a small vocabulary size with the ability to represent rare and out-of-vocabulary words. This allows BERT to effectively process a wide range of words, including rare and domain-specific words, while still maintaining a relatively small vocabulary size.

There are several advantages to using a vocabulary-based tokenization method like WordPiece. One advantage is that it can handle a wide range of words without requiring a large vocabulary size. This is particularly useful for pre-trained language models, which are trained on a large dataset and can be fine-tuned for a variety of tasks. Another advantage is that it allows the model to learn a more robust representation for rare and out-of-vocabulary words, which can improve performance on tasks that require handling these types of words.

However, there are also some disadvantages to using a vocabulary-based tokenization method. One disadvantage is that it can be more computationally expensive compared to other methods, as it requires maintaining a vocabulary and mapping words to subwords. In addition, there is the potential for errors in the tokenization process if the vocabulary is not sufficiently comprehensive or if the mapping of words to subwords is not done correctly.

Overall, WordPiece is a effective tokenization method for pre-trained language models, and has been widely used in systems such as BERT [14]. While it has some disadvantages, it has proven to be effective at handling rare and out-of-vocabulary words and has contributed to the success of pre-trained language models on a wide range of NLP tasks.

2.4 Zero-Shot and Few-Shot Learning

Zero-shot learning refers to the ability of a model to solve a task that it has not been explicitly trained on. Usually, the model is confronted with an explicit prompt, asked to solve a specific task or answer a specific question [9]. Likewise, models can be prompted the task to be solved together with a limited set of samples. This setting is called few-shot learning. The samples fed to the model can either be samples indicating the model how to solve a specific task, or, can be comprised of negative samples. This shows the model incorrect samples such that it does not get these wrong.

A simple example would be prompting "The sentiment of the movie review 'This movie is great' is _____". Many large PLMs would be able to solve this correctly by filling the gap with *positive* or *good*. PLMs like GPT-3 or XLnet have been shown to work well in a zero-shot setting [9, 77]. This can be largely explained by their sheer size and the

expressive power that comes with it. Apart from that, there is demonstrated work for BERT models in these types of settings, too [72, 67, 40]. These methods have proven to to be effective measures of employing PLMs' capabilities. For example, on SuperGLUE, GPT-3 is able to achieve almost-state-of the art results by just presenting a handful of examples.

Few-shot learning allows for more efficient training of machine learning models, as it requires fewer examples to learn a new task. Additionally, zero-shot learning allows for the training of machine learning models on tasks for which no labeled examples are available [9]. This can be especially useful in situations where it is difficult, expensive or impossible to obtain labeled data for a particular task. Due to that, both few-shot and zero-shot learning can be more efficient in terms of computation. No gradient updates are required, beyond those needed to train the base model. This makes the approach greener than always re-training large PLMs [49].

Few-shot learning can also improve the generalization ability of a model, as it is able to learn from a more diverse range of examples [63]. This can make the model more robust and able to perform well on a wider range of tasks. Apart from that, zero-shot learning can also improve the flexibility and adaptability of a model, as it is able to perform new tasks without the need for additional training data. This can make it more versatile and able to perform well on a wider range of tasks. Clearly, this can as well be a downside. PLMs in these settings are prone to which samples are provided and even factors like order of the samples can play a significant role [80]. Moreover, zero-shot learning is a great achievement in NLP, however, is prone to error. Models are highly sensitive to minimally differing prompts. In addition to that, often responses are retrieved which are grammatically correct and coherent, however, make no sense or are far off from what a human would answer.

One major disadvantage of zero-shot and few-shot learning is that it can be difficult to achieve good performance on complex tasks without any data or very few labeled examples. This is because the model must rely on its general knowledge and understanding of the world (or rather statistical propoerties between tokens it has seen during training) to perform the task, rather than being able to learn from explicit examples. While the setting works in many situations, it fails drastically in many as well. This can be especially true for more complex tasks, where a larger number of examples may be needed to learn the necessary relationships [66].

In the context of this work, the previously discussed advantages are not exactly the motivation of using the approach. Mainly, we employ few-shot and zero-shot learning to test the enhanced model's ability of certain concepts without fine-tuning (i.e., without changing the representations). Furthermore, we simply wanted to test BERT's few-shot and zero-shot capabilities and get a feeling of what is possible and what is not.

2.5 MMR

Mean Reciprocal Rank (MRR) is a widely used evaluation metric in the field of information retrieval and recommendation systems [73, 56]. It is a measure of the effectiveness of a ranking model and quantifies the quality of the returned ranked list of items for a set of queries. MRR is calculated as the average of the reciprocal ranks of the first relevant item in the set of ranked items. The reciprocal rank is the multiplicative inverse of the rank, which assigns a higher weight to higher-ranked results and a lower weight to lower-ranked results.

In the context of masked language models and cloze tasks, MRR can be used to evaluate the performance of a deep language model by measuring its ability to predict the missing word or phrase in a given context. The MRR of a language model can be calculated by taking the average of the reciprocal ranks of the correct answer across a set of test examples. A high MRR value indicates that the model is able to correctly predict the

missing word or phrase with a high degree of accuracy and at a high rank in the list of possible answers. This makes MRR a useful metric for evaluating the quality of language models and comparing the performance of different models on cloze tasks [21].

Mathematically, the MRR can be expressed as

$$MRR = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\text{rank}_i}$$

Where n is the number of queries and rank_i is the rank of the first relevant item for the i-th query. It is worth noting that in the above formula, the sum is over i which is the queries, so the formula calculates the reciprocal rank of the first relevant item of each query and take the mean of those ranks.

2.6 Related Work

There is a demonstrated body of work on improving deep pre-trained language models' abilities on performing well in domain specific situations. In the following, the most common approaches taken are elaborated.

2.6.1 Fine-tuning on a dataset specific to the domain

Though it is not explicitly designed for improving rare word understanding, fine-tuning a model on a dataset specific to the domain can help the model learn the specific terminology and concepts that are relevant to the domain. Further, it can improve its understanding of rare and domain-specific words [59], which may lead to significant performance gains on domain-specific tasks [79].

There is a whole body of work showcasing PLMs' capabilities of performing well on a range of **1.** specific tasks such as classification, named entity recognition or sentiment analysis [58, 60], and 2) domain specific applications such as in the medical, legal or financial domain [78, 57, 28].

There are, however, a few downsides to this. **1.** Fine-tuning requires a labeled dataset specific to the task or domain of interest. The model is trained on this labeled dataset in order to learn the relevant concepts and improve its performance on the specific task or domain. **2.** It is generally not able to perform well outside of the task it has been trained on resulting in limited capabilities for further use.

2.6.2 Using domain-specific language models

Instead of using a general-purpose language model like BERT or GPT-3, it is be possible to pre-train a domain-specific language model that is specifically tailored to the domain of interest. The most famous applications of this research direction compromise domain specific pre-trained language models such as BioBERT [27], Financial BERT [2, 32], Climatebert [68] among others.

While these models have a demonstrated history of performing better on a range of domain specific tasks they entail a range of serious defects. One main limitation is that it can be difficult to obtain a large enough dataset specific to a particular domain to train a high-quality domain-specific model. This can make it challenging to achieve the same level of performance as a general-purpose model that has been trained on a much larger dataset. Apart from data resources, in most scenarios it would not be feasible to train a very large PLM from scratch as it often requires days of computation on excessively expensive hardware [20].

Apart from that, models like BioBERT work well on general tasks compromising biological knowledge. But they likely wouldn't perform well on tasks which go down a level deeper, such as, molecular biology, genetics or pharmaceutics. Domain specific models do

work fairly well, but the range of possible domains which they would have to be adapted to is far too extensive to be covered by this approach alone.

2.6.3 Using domain-specific word embeddings

Word embeddings are a way of representing words as numerical vectors, and can be used to capture the meaning and context of a word [36]. Using domain-specific word embeddings, rather than general-purpose word embeddings, can help the model better understand rare and domain-specific words by providing better representations of concepts, thus improving associations, relations and understand of the world.

As this work is concerned with enhancing word embeddings of pre-trained language models, this section will go into detail about current progress in this field. The current default case of handling rare and domain specific words is assuming that a PLM's tokenizer splits up the word at hand into meaningful subunits. Tokenization algorithms such as byte-pair encoding [55] or WordPiece embeddings [74], do work really well in many situations. However, often this does not hold. Regard a word like *Deoxyribonucleic*. In genetics, this word is not a rare word at all. However, in day-to-day speech it is not used very often. BERT tokenizer WordPiece will break the word into the subtokens *de*, *##ox*, *##yr*, *##ib*, *##on*, *##uc*, *##lei*, *##c*. Clearly, breaking up the word into eight chunks carrying relatively low meaning will not improve the representation of *Deoxyribonucleic*.

One typical approach to addressing this issue is explicitly regarding the surface form of a word. This could be embedding words solely by its character n-grams [70], by morpheme information [34], enriching word vectors with subword information [8, 47] or using the context of a word to infer its meaning [22, 18]. Another way of achieving this is reached by recomputing (or mimicking) the representation of a word [43].

The Form-Context Model

The form-context model is a model for representing and learning embeddings for novel words, or words that did not appear in the training data [51]. The model leverages both the surface form of a novel word, such as its subword n-grams, and the context in which it occurs to learn an embedding for the word. The approach has been shown to result in large increases in embedding quality, achieving state-of-the-art results on the Definitional Nonce [18] and Contextual Rare Words datasets [22].

In principle, the approach looks at the surface form of the word (specifically, the unique characters) and as well as the context of it and constructs an average representation of subword information. For the surface form, a multi-set of character n-grams of the target word is constructed. Every n-gram in this multi-set is mapped to an embedding vector where the final surface form representation is just an average of these. For the context form, the representations of the context of the target word are averaged over a fixed window. Once obtained, the context and the surface representations are combined into a weighted representation [51].

Attentive Mimicking

The current paradigm for learning a word's representation by its context in natural language processing is to treat all contexts of a word by equal importance. Often only a few of a word's contexts provide valuable information about its meaning [52]. In "Attentive Mimicking: Better Word Embeddings by Attending to Informative Contexts", the authors propose a new approach, called attentive mimicking (AM), which uses self-attention to select a subset of especially informative and reliable contexts for a given word [52]. Although there have been approaches which leverage the informativeness of specific units, they do not look at choosing the right contexts, but the right words of a context [30]. AM is based on the observation that reliable contexts for a given word tend to resemble each other. The authors introduce the attentive mimicking model and demonstrate that it produces

high-quality embeddings for rare and medium-frequency words. They also propose a novel evaluation method based on VecMap to easily evaluate the embedding quality of low- and medium-frequency words. The results of their evaluation show that attentive mimicking improves word embeddings on various datasets.

As previously explained, the form-context model utilizes the surface form as well as contexts of a given target word. The main issue with this approach is that only a fraction of the contexts at hand are actually useful and informative. To address this issue, the AM model, in essence, adds a weighting parameter to the respective contexts and lets the model decide which are informative and which are not. In addition to that, AM weights contexts by its similarity. The assumption at hand is that the contexts, in which the word mainly occurs, correspond to each other and are very similar.

This coincides with the observations made in Section 3.2. Even if tens of definitions are available, only by qualitatively investigating one observation is salient. Most definitions of a word are very specific and correspond to a niche the word occurs. Very few definitions cover the vast majority of the word's meaning. Clearly, one could argue that there *are* going to be situations where these definitions could be of use. But generally speaking, it seems that the model performs better if general concepts are presented to it.

WordNet Language Model Probing (WNLaMPro)

To explicitly test whether representations of rare words in PLMs have improved, the WordNet Language Model Probing (WNLaMPro) dataset was introduced [48]. In principal, the dataset tests the model's ability to solve the cloze task it was originally trained on. Most current PLMs do not have any issues filling in the gap of the sample "A lime is a _____" where possible valid answers could be lime, lemon or fruit. However, asking an ordinary PLM to fill the gap in "A kumquat is a _____" is an extremely challenging task and most of the time does not yield the correct response.

The WNLaMPro dataset is comprised of four types of relations to be tested, that is, antonyms (ANT), hypernyms (HYP), cohyponyms (COH) and corrupted (COR) samples. The dataset contains a set of samples (k, r, T) where k is a keyword, r is a relation and T is a set of target words. Moreover, for each of the four relations, the dataset provides a set of patterns. For example, a pattern for an relation of type (ANT), for example, could be "¡W¿ is the opposite of _____", where $< W >$ is an arbitrary keyword. The exact demonstration for each relation's patterns is omitted for the sake of conciseness and can be retrieved from the previously cited paper.

The dataset was split into a development set and a test set, with 10% of all entries randomly selected for the development set and the remaining 90% forming the test set. There are three subsets defined based on keyword counts: WNLaMPro-RARE (occurring less than 10 times), WNLaMPro-MEDIUM (occurring 10 or more times but less than 100 times), and WNLaMPro-FREQUENT (all remaining words). Further, the samples are split into three parts. **1.** $(0, 10)$, which correspond to the most frequent words **2.** $(10, 100)$, which are medium-frequency words, and **3.** $(100, -1)$, being rare words.

ArXiv Document Classification Dataset

In addition to the WNLaMPro evaluation we test Defbert on the ArXiv dataset [13]. The ArXiv dataset is a multipurpose dataset containing of 1.5 million pre-print articles collected over a span of 28 years. The dataset is split into the three domains physics, mathematics and computer science. For our purpose we regard a classification task but it can be used for a variety of tasks.

As mentioned before, the dataset consists of three domains, namely, physics, mathematics and computer science. Each domain has a small dataset with 1200 (called 1+e03) and large dataset with 12000 (called 1+e03) samples. Each dataset is divided into a train, validation and test split with ratios of 0.6, 0.2 and 0.2. The dataset consists of titles of

research papers which shall be mapped to one of 20 classes. For example, the title *automation of road intersections using consensus-based auction algorithms* shall be mapped to the class *systems and control*.

Clearly, this ratio can be shifted arbitrarily. To test Defbert capabilities, an ordinary BERT model without new embeddings, as well as a a Defbert model are fine-tuned on the text classification task (see Section 3.5.2). Moreover, we could even use almost 100% of the samples since we use few-shot and zero-shot learning to test the model (see Section 2.4). There is no need to for a large training set (if at all).

3 Defbert

In this chapter the principle idea of **Defbert** is explained. Defbert addresses the issue of low-quality representations of infrequent words in PLMs' embedding spaces. The core idea is to fine-tune an arbitrary PLM on a dataset of simple definitions. The task at hand is the following. Given a set of definitions of a one-token-word (OTW) the model shall infer its true embedding. Once trained, representations for unseen words are to be inferred and can be readily used for other downstream tasks. Principally, words which do not have single-token-representation are the subjects of interest here.

The idea is in depth explained in Section 3.1. The definitions used as well as its variations are shown in Section 3.2. In addition to definitions, in Section 3.3 I shed light on the patterns used in training. Further, the exact training procedure with all its specifications are shown in Section 3.4. Lastly, the approach is evaluated and discussed in Sections 3.5 and 3.6, respectively.

3.1 Approach

We regard a PLM and fine-tune it on a custom task. In this work, OTWs and OTR are practically interchangeable synonyms, however, differ sightly depending on what is being explained. Clearly, claiming a word is a OTW it follows that it has a OTR. The exact opposite situation holds as well. However, it may make sense to use specific terminology in specific situations. Simply speaking, the term OTW is used if we specifically regard the word as a unit (still implying that is has a single-token-representation). The training data at hand is predetermined through the set of OTWs in the model's tokenizer. The model's tokenizer does not know of representations, thus OTW is the correct term to use here. For all situations where we specifically talk about a OTW's representation OTR is used.

For each of the OTW the task is to infer its embedding given a set of its definitions. These definitions are scraped from various sources like Wiktionary and WordNet. Once the definitions are obtained, they are formatted into a specific pattern for the model to be trained on. As the OTR of these words are known a priori we use these as ground truth to train the model. Once the model is trained we can ask it to infer new OTR of words which originally would be split up by the model's tokenizer.

3.2 Definitions

As our dataset for the training we scrape definitions from various sources. We decide that Wiktionary and WordNet shall serve as our main source as they largely cover most OTRs.

For BERT, out of the 30522 tokens in its tokenizer 23694 are OTRs, which account for 77.6%. For RoBERTa, out of the 50265 tokens in its tokenizer 33135 are OTRs, which account for 65.9%. We can see that relatively speaking, BERT has more OTRs than RoBERTa. Absolutely speaking, RoBERTa has around 10.000 more training samples available. This indicates that RoBERTa might be the better backbone model as the limit for the amount of data available is higher. Originally, I was going to test BERT's *and* RoBERTa's capabilities, hence the numbers in this section. However, due to time constraint, ideas were only tested on BERT.

WordNet vs. Wiktionary Definitions

WordNet is known for its fine-grained senses [17]. This becomes visible when taking a closer look at its definitions. When examining the data closer, it becomes evident that Wiktionary, too, provides a wide range of definitions. In the following I will present a qualitative analysis of the definition of *green*. I will showcase what definitions look like, point to peculiarities and discuss a range of problems and design choices to make when using the data.

Word Classes

Both WordNet and Wiktionary provide definitions for nouns, verbs, adjectives, adverbs and proper nouns. For a specific lexem (i.e., how the word is written) there exist various lexical units (i.e., the meaning of the word) [31]. Therefore, it is only natural that when regarding a particular lexem any reliable source will yield multiple definitions. Clearly so, this is the case for both Wiktionary and WordNet.

In the following, I analyze the the word *green*, seperated into its respective word classes.

Verbs

The first class we analyze is the class of *verbs*. Verbs are an interesting class of words to examine more close up since they come with many inflectional variations. For example, regard the verb *swim*. It may be realized as *swam*, *swum*, *swims* and *swimming*. All realizations describe exactly the same concept, *swimming*, but infuse more knowledge about the tense and the person described performing the action. In this particular case, realizations are likely to be encountered frequently. But if we regarded a rare verb, the same sense may be expressed in various lexems (disregarding the tense). This may result in the model originally understanding the verb well but not other forms of it.

WordNet provides one definition for its verb realization, that is:

1. turn or become green

Wiktionary, on the other hand,. provides $five$ definitions for its verb realization:

1. (transitive) To make (something) green, to turn (something) green.
2. To become or grow green in colour.
3. (transitive) To add greenspaces to (a town, etc.).
4. (intransitive) To become environmentally aware.
5. (transitive) To make (something) environmentally friendly.

Not that, in this specific case, rarely ever the word *green* will be used as a verb. Most of the time, to express I want to turn something green or greener one would use *"I am making / turning this green"*. Already here it becomes clear that a wide range of definitions are provided, whose usefulness is questionable.

In the examples above, WordNet definition 1 (WN1) as well as Wiktionary definition 1 (WK1) are likely the senses used in the majority of the cases (if used at all). WK2 and WK3 are used occasionally, but likely are not the correct sense in many situations. There is a high chance definitions WK4 and WK5 are rarely ever used. For example, if one wanted to express a company is becoming greener most likely one would not claim *"they are greening"* but say *"they are becoming greener"*. Clearly, this is just an example and the explanations and reasoning given above is solely based on intuition. No deep numerical analysis was conducted. But it shall showcase the carefulness one has to conduct when employing this data source.

In this example, WordNet provides far less definitions for the verb form of the word *green* than Wiktionary does. On average, for every verb in WordNet, we get 0.65 definitions in Wiktionary.

Nouns

Wiktionary provides a considerably higher amount of definitions for the noun version of *green* than WordNet does. In the case below, WordNet only yields 7 definitions whereas Wiktionary provides 14. Generally speaking, on average, for every noun in WordNet, we get 1.3 definitions in Wiktionary. In the case of nouns, we can see that from a quantitative perspective Wiktionary is the richer source. Qualitatively, it is hard to tell which serves the purpose better. A higher number of definitions does not necessarily indicate better quality. It could also mean that the definitions provide contexts which are only used in specific niches. Possibly, these definitions do not contribute much to a general strong understanding of a word.

WordNet provides 8 definitions for the noun realization of the word *green*, that is:

1. green color or pigment; resembling the color of growing grass;
2. a piece of open land for recreational use in an urban area;
3. an area of closely cropped grass surrounding the hole on a golf course;
4. United States labor leader who was president of the American Federation of Labor from 1924 to 1952 and who led the struggle with the Congress of Industrial Organizations (1873-1952);
5. an environmentalist who belongs to the Green Party; river that rises in western Wyoming and flows southward through Utah to a tributary of the Colorado River;
6. street names for ketamine;
7. any of various leafy plants or their leaves and stems eaten as vegetables.

In Wiktionary, for the noun realization of *green*, 14 definitions are provided, that is:

1. The colour of growing foliage, as well as other plant cells containing chlorophyll; the colour between yellow and blue in the visible spectrum; one of the primary additive colour for transmitted light; the colour obtained by subtracting red and blue from white light using cyan and yellow filters.
2. (politics, sometimes capitalised) A member of a green party; an environmentalist.
3. (golf) A putting green, the part of a golf course near the hole.
4. (bowls) The surface upon which bowls is played.
5. (snooker) One of the colour balls used in snooker, with a value of 3 points.
6. (Britain) a public patch of land in the middle of a settlement.
7. A grassy plain; a piece of ground covered with verdant herbage.
8. (chiefly in the plural) Fresh leaves or branches of trees or other plants; wreaths.
9. Any substance or pigment of a green colour.
10. A green light used as a signal.
11. (uncountable, slang) Marijuana.
12. (US, slang, uncountable) Money.
13. (particle physics) One of the three color charges for quarks.
14. (theater, informal) Short for green room.

Having a look at the WordNet definitions, WN1 to WN3 likely provide the correct context in most situations. WN4 to WN6 are rarely ever going to be the correct description of a word. They describe extremely specific situations talking about politics and drugs. WN7 is only useful in very specific cases.

Having a look at the Wiktionary definitions one thing specifically catches the eye. WK1 is comprised of a broad range of various definitions packed into one. While this may be helpful as the most important concepts of *green* are explained in one go, it could also break the pattern which the model sees most of the time.

Importantly, notice how most definitions are focused on a very fine-grained concept. *Politics*, *golf*, *bowls*, *snooker*, *physics*, all of which *green* is used in. Though these are senses which are used extremely seldom in day-to-day language in those specific situations they do occur frequently. Other than that, the tag appended to the front of the samples exactly

provides the model context as to where this specific definition applies. One might think that due to the extra information Wiktionary yields as a better source for model training. But ss can be seen in Section 3.5.1 Wiktionary definitions yield a worse performance.

Adjectives

Analyzing the definitions of the adjective realization of *green* one can see that WordNet provides 5 definitions whereas and Wiktionary provides 14. Again, Wiktionary is the much richer source and as before the extra information appended to the front of a sample may be a helpful indication for the model.

In WordNet, 5 definitions are given for the adjective form of the word *green*, being:

1. of the color between blue and yellow in the color spectrum; similar to the color of fresh grass,
2. not fully developed or mature; not ripe,
3. looking pale and unhealthy,
4. naive and easily deceived or tricked,
5. concerned with or supporting or in conformity with the political principles of the Green Party,

For the Wiktionary definitions, we obtain:

1. Having green as its color.
2. (figuratively, of people) Sickly, unwell.
3. Unripe, said of certain fruits that change color when they ripen.
4. (figuratively) Inexperienced.
5. (figuratively) Full of life and vigour; fresh and vigorous; new; recent.
6. (figuratively, of people) Naive or unaware of obvious facts.
7. (figuratively, of people) Overcome with envy.'
8. (figuratively) Environmentally friendly.'
9. (cricket) Describing a pitch which, even if there is no visible grass, still contains a significant amount of moisture.'
10. (dated) Of bacon or similar small goods: unprocessed, raw, unsmoked; not smoked or spiced.'
11. (dated) Not fully roasted; half raw.'
12. (film, television, historical) Of film: freshly processed by the laboratory and not yet fully physically hardened.'
13. Of freshly cut wood or lumber that has not been dried: containing moisture and therefore relatively more flexible or springy.'
14. (wine) High or too high in acidity.'
15. (Philippines) Having a sexual connotation.'
16. (particle physics) Having a color charge of green.'
17. Being or relating to the green currencies of the European Union.'

Extra Information

There are a few things to note about the definitions at hand. As discussed previously, the granularity of definitions differs substantially. However, since extra information is added to front I assume this assists the model to get a better grasp of the overall concept. Information provided can be literal contexts such as *wine*, *Philippines*, *Math* or *Football* but can be provide additional information such as *outdated*, *obsolete* or *figuratively*. Intuitively, the additional information should help the model make perform better as more information is more evidence. However, as can be seen in Section 3.5.1 Wiktionary definitions perform significantly worse than WordNet definitions. Wordnet does seems like the better source to proceed with.

Self-reference

An issue to consider are self-references. This may occur in three ways. One, the target word itself is contained in the definition and poses the exact same sense and word class. Two, the target word itself is contained in the definition but may have a different sense or may be from a different word class. E.g, the progressive form of the verb *running* may be defined as *the act of running*, where both *running*s are affiliated with different word classes. Or, three, a different form of the target word is contained in the definition. E.g, the infinitive form of the verb *run* may be defined as *the act of running*. Intuitively, it is hard to understand if these types of definitions influence the outcome of the approach and and if yes, in which order of magnitude. Looking at the amount of times this issue occurs we obtain the following results. In both sources these numbers roughly correspond. 10% of all samples contain an identical lexem as the target word itself. 25% of samples contain a word highly related to the target word (analyzed through stemming all words in the definitions). As results in 3.5.1 show including or excluding samples with self-references unexpectedly barely make a difference.

Proper Nouns, Niche Information and Obsolete Senses

Lastly, proper nouns, niche information and obsolete senses need to be regarded. It is hard to tell how many of the samples regard niche information, and which of the definitions provide the correct sense in most of the situations. Likely, only a small portion of definitions supply the correct description (similar to the Pareto principle) [48]. For proper nouns and obsolete senses the amount of samples can be counted in Wiktionary. Of all definitions, around 5% constitute for obsolete senses and 0.5% are proper nouns. Likely, this is the reason why it barely makes a difference if these are systematically excluded or not. This, however, does not mean that being selective in the definitions to use generally does not make a difference. Overall, the magnitude of this effect is hard to determine without further examination.

3.3 Patterns

The patterns discussed in this section, in principle define how to provide the information fed to the model. In the way we define the pattern we have to regard various design choices. Either samples are fed in a multi-definition or single-definition pattern style. In the multi-definition approach for a target word all definitions are appended into a single sample which the model learns from. The target word with all definitions is provided at once such that only a single gradient update is conducted for an entire set of definitions. In the single-definition approach for a target word definitions are provided one by one. For each target word, we conduct a range of gradient updates which the model accounts for separately.

Multi-Definition Pattern

Firstly, we have a look at the multi-definition pattern. The pattern for this setting may look like this, but does not necessarily have to: *The word [MASK] is defined by the following definitions: 1. [DEF-1], 2. [DEF-2], ...*, where *[MASK]* is the word to be inferred and *[DEF-1]* to *[DEF-N]* are definitions as shown above.

This approach has the benefit that it is **1.** computationally more efficient as only one sample per target word is provided. Besides the maximal token length is much more made use of. Intuitively, one could regard the provision of all samples at once as a measure of smoothing as batch sizes are increased [29]. Intuitively speaking, a single definition will not have the ability to influence the resulting representation heavily. Many definitions are in place influencing gradient updates not allowing a single one to distort the representation into a specific direction which is possibly far off from what the others (or the best)

directions would be.; **2.** the model can make sense of possible relations between the senses; **3.** when performing inference, we do not need to select which is the correct definition to feed to the model. Simply all definitions at once are provided. While the last point may be regarded as a positive aspect it might also be a problem. If during inference we only provided the single correct definition, results might be much more accurate. However, since the words of interest are mostly infrequent words they often only have very few senses such that this may not be a problem. The overall low-level difference between the approaches is hard to evaluate without further investigation (not conducted due to time constraints).

Single-Definition Pattern

For the single-definition pattern we would create N samples and feed them to the model one by one. One example of what such a pattern could look like would be *The word [MASK] is defined by the definition [DEF-i].* The model would see each target word as many times as definitions are available for it. As described in the previous section, this would result in a higher computational complexity, more specific (and possible noisy) gradient updates and would have to make a decision as to which definition of the new word of interest to feed to the model.

The single-definition approach might be the more effective one due to the following reason. The words of interest during inference are rare and domain specific words. The example show above (the word *green*) is a highly ambiguous word which can be used in a range of situations. Likely, it is valid to assume that for words of interest not many different definitions are available. Often these words are very specific and only applied to one unique situating. Using the single-definition approach would alleviate the mismatch between training data and samples to be inferred. Often we provide training examples with 100, 200 or even close to the maximum of 512 tokens. During inference, often we only have 1 or 2 definitions. If the single-definition approach was conducted, despite regarding rare words in many cases we would have to decide which is the correct definition to infer the embedding from.

3.4 Training

The training procedure is relatively straightforward. In the following, the exact setting is reported. The model in use is huggingface's bert-base implementation. Since the actual training is just a fine-tuning procedure, it is possible to experiment on a single GPU. The framework used to train the model is pytorch-lightning. Since pytorch-lightnings implementation is extremely lightweight and quickly adaptable, training on multiple GPU, if available, should not pose a problem. However, for this setting it wasn't necessary as training is conducted quickly.

Data

Data is scraped beforehand to **1.** decrease compute time during training and, **2.** reuse the data repeatedly. To achieve this, I use a web scraper provided by my supervisor Lütfi Kerem Senel. Once the data is obtained, it is split into train, test and validation sets according to a 80-10-10 split.

Hyperparameters

As an optimizer, I use AdamW [33] with a constant learning rate of $5e-5$, β_1 is set to 0.9 and $beta_2$ is set to 0.999. Further, ϵ is equal to $1e-8$. I train with a batch size of 12, though this is dependent on the GPU in usage. Likely, a higher batch size results in better convergence. The maximum sequence length is set to 256 even though the model

is capable of handling sequences of up to 512 tokens. This is mainly due to the fact that setting it to a lower value increases computational efficiency. Only a small fraction of samples is being cut by the restriction, hence this design choice.

3.4.1 Ordinary Defbert Setting

Ordinary Defbert, which I generally refer to as Defbert, is trained on all of its tokenizer's OTWs. For example, a training sample for the word fish will be constructed as "The word [MASK] is defined by the following definitions: 1. (countable) A cold-blooded vertebrate animal that lives in water, moving with the help of fins and breathing with gills. 2. (archaic or loosely) Any animal (or any vertebrate) that lives exclusively in water. 3. (Newfoundland) Cod; codfish. 4. ...". The token "[MASK]" is then to be predicted and will be compared to the true embedding. The prediction and the ground truth are compared by the means of cosine similarity, thus our objective to minimize becomes $L(e, \hat{e}) = 1 - \frac{e \cdot \hat{e}}{||e|| \cdot ||\hat{e}||}$ where e is the true embedding of a target word and $\hat{e}$ is the predicted embedding.

Embedding Inference

Once Defbert is trained it can be used to infer new OTRs for an arbitrary target word. Importantly though, the target word needs to have a definition. In the best case this definition is extracted from Wiktionary or WordNet, such that the form of the definition resemble that of the training data. If not, the model should still perform reasonably well. It can happen though, that performance deviates from test performance due to a possible out of domain structure of new samples, however, this is a general problem in machine learning.

A new set of words can then simply be inferred, e.g., iterating over an arbitrary target data set. One has to search for words which the original model tokenizer would break up into chunks. Likely, these words are rare or domain specific words. For these words, we infer the new embedding and add it to the model as well as its tokenizer. Once done, the new model is ready to be used.

3.4.2 Hierarchical Defbert

As the ordinary Defbert approach does not yield the results hoped for I conducted further experiments aiming to create better embeddings using an approach called **hierarchical Defbert**. Importantly to note, this approach does not affect the training procedure but only the model usage and its inference of new embeddings.

Assume the following simple example. During training, most of the tokens at hand describe very simple concepts such as *dog*, *apple* or *building*. When training these, the words used to describe these concepts are comparably simple. Having a *trained Defbert model* at hand, let the target word be w of which we scrape the definitions. The target word w is an infrequent or domain specific word. There is a high chance that, unlike what has been seen during training, the definitions of w itself are rather complicated or mention rare words. Originally, the approach tried to infer OTRs of infrequent words, however, if the words used to infer these words themselves are rare we are back to our original problem.

Thus, once encountering a definition with rare words itself we infer these first. This idea can be extended arbitrarily in depth. If you take a look at Figure 3.1 you can see the conceptual strategy. Let's assume we try to infer the embedding for *eccentrically*. One of the definitions is *not symmetrically with respect to the center*. Now, if we wanted to infer an embedding for *eccentrically* using Defbert, *symmetrically* itself may not have a good representation due to it being underrepresented. Thus, we infer a new embedding for *symmetrically* first. We continue by obtaining *symmetrically*'s definition and obtain the rare word *corresponding*. Since we have reached our pre-defined maximum depth of 2,

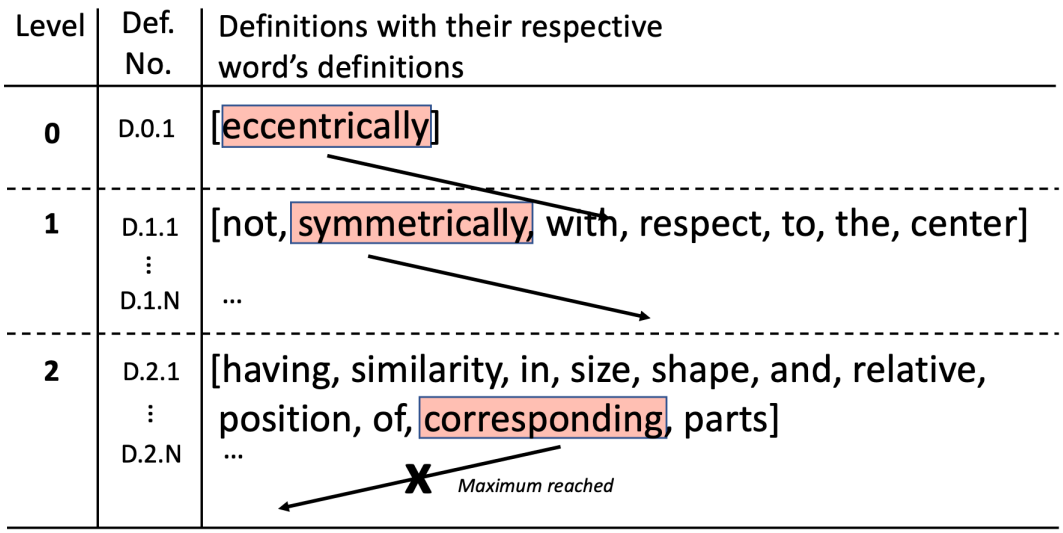

Figure 3.1: In $D.0.1$ (= definition 1 at level 0) we try to infer eccentrically. All words that are marked red are regarded as rare. First, we obtain the definition of *eccentric*, then we infer an embedding for *symmetrically* as we choose not go deeper than 2 levels.

we do not go deeper and leave *corresponding* as is and use the original tokenizer's subword representation.

This idea has two design choices to make. **1.** we have to decide for a threshold when a rare word is regarded as rare word, i.e., for which words do we assume the tokenizer's multi-token-representation is not sufficient. It can happen that words are broken up into subwords, for example, assume *cats* to be broken up into *cat* and *##s*. Likely, the model would not have any problems understanding that the *s* is simply the plural suffix of *cat*. Thus, it can happen that words are indeed broken up, but from an information perspective does not pose a problem in understanding. **2.** we need to decide how deep do we go down the definition tree, as shown in Figure 3.1. The original Defbert model converges at around 0.3. In simple terms that means the newly inferred embeddings are on average 70% of its original quality. Thus, we can expect that the deeper we go, the more quality of new embeddings suffer. Likely, going down only 2 or 3 levels yields the best results.

Once these parameters are chosen the actual implementation is fairly straightforward. Construct a *definition tree* with the depth of the chosen maximum level m. This is done as following. For a target word w, obtain all definitions. For each word in these definitions, check whether it contains other words below a certain threshold τ. As a word count indicator we count words in Wikipedia and assume it represents the overall word distribution well. For each word smaller than τ, again, obtain its definition. Repeat this until m is reached. Save this in a tree structure as shown in Figure 3.1.

Once the *definition trees* are constructed we can start making use of the structures. The idea is simple. We start inferring embeddings bottom-up from the lowest level to the top. For each level, we infer the embeddings and add them to the model. For example, if we are in level m we infer embeddings. Every embedding is added to the tokenizer and model. Moving to level $m-1$, all words below τ now have a new, known OTR which is ready to use. Now, for level $m-1$ we infer new embeddings and add them to the model. This procedure is repeated until level 0 is reached. Reaching the top, we disregard all multi token representations below a certain threshold and make use of our new inferred embeddings.

As mentioned before, the idea has a major downfall we hoped would not have a big impact. When training Defbert, the training loss approximately converges towards 0.3. In other words, on average, when approximating a representation by its definition we achieve

a similarity of 70%. This might still be sufficient if the the multi token representation is worse. But, this means that every time we go down the definition tree one step we lose, on average, 30% accuracy in its representation. As this is just a rough estimate, it is hard to tell what the impact of this is. But likely, the deeper we go, the less accurate representations become.

3.5 Evaluation

In this section Defbert is evaluated. We evaluate the approach in various types of settings. Firstly, Defbert is evaluated on WNLaMPro in Section 3.5.1, next results on the ArXiv document classification task are shown in Section 3.5.2. Further, we test the hierarchical Defbert approach on WNLaMPro in Section 3.5.3. Lastly, we will have a closer look at Defberts capabilities in few-shot and zero-short setting in Sections 3.5.4 and 3.5.5.

3.5.1 Defbert on WNLaMPro

Results shown in this section are based on an evaluation with the WNLaMPro dataset. A detailed explanation of the dataset can be found in Section **??**. The model tested here is BERT. To generate insightful statistics, only samples in WNLaMPro which do not have a OTR in the model's tokenizer are tested. Out of the around 30.000 tokens in BERT's this accounts for around 17.000 examples. The definitions used in this experiment are solely based on WordNet definitions. In Table 3.1 one can observe the results obtained. The experiments conducted in this subsection are based on preliminary experiments by my supervisor Lütfi Kerem Senel. The results are therefore based on his efforts.

	Count	BERT	AM	Defbert0	Defbert1	Defbert2	Defbert3
ALL	17028	0.295	-	0.431	0.427	0.432	0.429
ALL (0,10)	2867	0.195	-	0.425	0.424	0.426	0.420
ALL (10,100)	4136	0.234	-	0.423	0.424	0.432	0.424
ALL (100,-1)	10025	0.348	-	0.436	0.429	0.433	0.433
ANT (0,10)	36	0.149	0.449	0.422	0.407	0.463	0.351
ANT (10,100)	53	0.089	0.511	0.474	0.455	0.438	0.410
ANT (100,-1)	241	0.390	0.482	0.355	0.371	0.396	0.354
HYP (0,10)	1065	0.276	0.300	0.540	0.529	0.529	0.538
HYP (10,100)	1602	0.327	0.343	0.562	0.563	0.562	0.569
HYP (100,-1)	4287	0.416	0.377	0.565	0.553	0.554	0.565
COH (0,10)	1766	0.147	0.213	0.356	0.361	0.363	0.350
COH (10,100)	2481	0.177	0.213	0.332	0.334	0.348	0.330
COH (100,-1)	5497	0.294	0.262	0.339	0.336	0.340	0.334

Table 3.1: Comparison between BERT, Attentive Mimicking (AM) and Defbert trained on various subsets of available definitions in WordNet. The metric at hand is the mean reciprocal rank (MRR). ALL regards all subtasks of the WNLaMPro dataset, ANT means antonyms, HYP means hyponyms and HYP means cohyponyms.

Here, six models are compared to each other. BERT serves as a baseline and the results of the Attentive Mimicking (AM) approach are used to judge the effectiveness of the approach. The difference between Defbert0 to Defbert3 in principle is the types of definitions used during training. Defbert0 disregards all proper nouns, self-references and obsolete senses. Defbert1 disregards all proper nouns and obsolete senses. Defbert2 only disregards proper nouns and Defbert3 keeps all three in the selection of definitions.

As can be seen, BERT does not constitute a strong baseline and is easily beaten by all four Defbert models across all subsets of WNLaMPro. For the antonyms, the AM

approach is the best across the board. There are no results across all sub-tasks for AM as the approach is not re-implemented and numbers are just taken from the paper. The authors do not report the numbers for the overall dataset. Comparing across the Defbert models, we can see that overall Defbert2 performs best. However, differences between the models only vary marginally. This indicates that the detailed quality of the chosen definitions may not be as relevant as previously assumed.

Moreover, the quality between Wiktionary and WordNet definitions are compared to each other. Thus in 3.2 one can observe the results if Defbert is trained on Wiktionary definitions. Here, around 15.000 samples contribute to the effective sample count.

	Count	Bert	Defbert-0	Defbert-1	Defbert-2	Defbert3
ALL	15180	0.309	0.356	0.353	0.356	0.347
ALL (0,10)	1879	0.210	0.241	0.241	0.237	0.237
ALL_M (10,100)	3718	0.242	0.325	0.334	0.339	0.318
ALL (100,-1)	9584	0.354	0.390	0.382	0.385	0.380
ANT (0,10)	35	0.153	0.493	0.635	0.604	0.514
ANT (10,100)	51	0.092	0.571	0.648	0.625	0.596
ANT (100,-1)	234	0.386	0.444	0.448	0.450	0.444
HYP (0,10)	669	0.298	0.323	0.313	0.305	0.307
HYP (10,100)	1428	0.338	0.435	0.432	0.435	0.420
HYP (100,-1)	4064	0.422	0.477	0.453	0.459	0.459
COH (0,10)	1175	0.161	0.187	0.189	0.188	0.188
COH (10,100)	2239	0.185	0.250	0.265	0.270	0.246
COH (100,-1)	5285	0.301	0.322	0.325	0.325	0.315

Table 3.2: WNLaMPro Results of BERT and all Defbert variations trained on Wiktionary dictionary. What can be seen is that across the board, models perform worse on these types of definitions despite expecting that the prepended information may be informative to the model.

What can be seen is that the overall MRR across all variations significantly decreases when using Wiktionary. This leads to the assumption that WordNet definitions are a lot better suited for training models of this type. Thus, all further investigations are conducted using Defbert2 trained on WordNet definitions, as it performed the best out of all tests. This result is a bit surprising since, as seen in Section 3.2, we saw that Wiktionary definitions are **1.** much richer in quantity and, **2.** richer in quality (presumably). Extra information is appended to the front of the samples. Apparently this does not help the model a lot. Possibly the fine granularity of Wiktionary (though WordNet, too, is quite fine-grained) hinders the model to learn well.

While these results show promising numbers they are not a sufficient indication for the new embeddings' quality. Testing BERT on non-downstream tasks only is known for its lack of expressiveness, only downstream tasks themselves are good indicators for a language model's capabilities [11]. This, however, posses a serious challenge for this work. If we evaluated the new embeddings on ordinary tasks such as sentiment analysis, question answering or summarizing, only a fraction of the words within these tasks would intersect with the newly inferred words. Unfortunately, there is a shortage of tasks and datasets to test rare word abilities of PLMs. Nevertheless, in the following we construct further test to examine whether the model has improved.

3.5.2 Defbert on Scientific Text Classification

In this setting we test DefBert on the ArXiv Scientific Document classification task. A thorough description of the dataset can be found in Section 2.6.3. Various sizes of training partitions and domains are tested to understand how training set size and domain affect

Dataset	Train Size	Bert	Defbert2	Difference
maths_1e+02	300	35.5	34.25	-1.25
	600	43.25	42.2	-1.05
	900	48.35	45.2	-3.15
	1200	49.05	46.7	-2.35
cs_1e+02	300	39.1	38.65	-0.45
	600	45.65	45.25	-0.4
	900	47.4	47.05	-0.35
	1200	49.55	50.25	0.7
physics_1e+02	300	40.95	40.4	-0.55
	600	52.7	50.3	-2.4
	900	58.85	56.35	-2.5
	1200	59.9	59	-0.9

Table 3.3: Comparison between BERT and Defbert fine-tuned on the ArXiv document classification task. The model's capabilities are compared across all three domains, namely, mathematics (maths), computer science (cs) and physics. Reported results are averaged over 5 runs.

the model's capacity. Results can be seen in Table 3.3. The experiments conducted in this subsection are based on preliminary experiments by my supervisor Lütfi Kerem Senel. The following results are therefore based on his efforts.

For the ArXiv document classification, the results shown in table:preliminaryresults3 clearly indicate that the Defbert approach does not work very well. In all three domains (maths, computer science and physics) Defbert2 trained on WordNet performs significantly worse than ordinary BERT.

3.5.3 Hierarchical Defbert on WNLaMPro

To evaluate the capabilities of the hierarchical Defbert approach (explained in Section 3.4.2) just like ordinary BERT we test the model on the WNLaMPro dataset.

τ	**BERT**	**AM**	**DB2**	7.5K	7.5K	7.5K	10K	10K	10K	12.5K	12.5K	12.5	15K	15K
m				1	2	3	1	2	3	1	2	3	1	2
ALL	0.295	-	0.432	0.296	0.294	0.295	0.278	0.277	0.277	0.275	0.273	0.294	0.276	0.272
ALL_F	0.195	-	0.426	0.197	0.197	0.198	0.197	0.197	0.197	0.196	0.196	0.194	0.196	0.195
ALL_M	0.234	-	0.432	0.238	0.235	0.237	0.229	0.228	0.229	0.230	0.228	0.231	0.232	0.230
ALL_R	0.348	-	0.433	0.348	0.346	0.346	0.322	0.321	0.320	0.317	0.314	0.349	0.316	0.312
ANT_F	0.149	0.449	0.463	0.157	0.157	0.157	0.157	0.157	0.157	0.141	0.141	0.133	0.141	0.141
ANT_M	0.089	0.511	0.438	0.136	0.126	0.126	0.099	0.097	0.097	0.100	0.097	0.098	0.111	0.115
ANT_R	0.390	0.482	0.396	0.395	0.390	0.389	0.340	0.340	0.339	0.326	0.326	0.385	0.320	0.313
HYP_F	0.276	0.300	0.529	0.278	0.277	0.278	0.278	0.278	0.278	0.278	0.277	0.275	0.277	0.276
HYP_M	0.327	0.343	0.562	0.322	0.320	0.322	0.313	0.314	0.314	0.313	0.312	0.321	0.315	0.313
HYP_R	0.416	0.377	0.554	0.420	0.419	0.419	0.393	0.392	0.392	0.388	0.388	0.415	0.390	0.384
COH_F	0.147	0.213	0.363	0.148	0.149	0.150	0.148	0.148	0.149	0.148	0.148	0.146	0.148	0.147
COH_M	0.177	0.213	0.348	0.185	0.182	0.184	0.178	0.175	0.177	0.178	0.176	0.176	0.180	0.178
COH_R	0.294	0.262	0.340	0.290	0.288	0.288	0.267	0.264	0.264	0.261	0.257	0.297	0.259	0.256

Table 3.4: Comparison between BERT and Hierarchical Defbert2 tested on the WNLaMPro dataset. F indicates the most frequent words, M indicates words with medium frequency and R indicates rare words. τ is the threshold of when to regard a word as rare and m indicates the depth of the decision tree to build. DB2 is Defbert2 and AM means classic Attentive Mimicking [48].

As can be seen in Table 3.4, the hierarchical Defbert approach performs worse than Defbert2 across all relation types. The performance drop is high enough to be almost on par with the BERT baseline again. Due to these results we decide not to further investigate the hierarchical approach and stick to ordinary Defbert.

3.5.4 Defbert's Few-Shot Abilities: Binary Classification

There are two problems evaluating Defbert. **1.** many downstream tasks do not contain enough rare or domain specific words to conclude expressive results. Likely, the performance will remain unchanged. **2.** fine-tuning the model to a specific task would change the internal state of the PLM. The goal for this work, however, is to obtain a PLM which is not fine-tuned to a specific task. We want to obtain a new PLM enhanced by the new representations which can be fine-tuned afterwards. To overcome these issues we construct new tasks based on few-shot and zero-shot-learning.

The objective of this evaluation is solving the ArXiv document classification task without explicitly fine-tuning the model to do so. In our first setting we construct a task similar to one-vs.-rest binary classification [19]. We ask the model whether a specific document belongs to the class and decide for the correct class based on the most certain *Yes*. In the second setting we compute the probabilities of the exact name of the classes and opt for the highest.

The binary classification task is constructed as following. Given a paper of a title t out of the dataset $\mathcal{T}$ we want to know which class c out of the possible classes $\mathcal{C}$ is the correct one. An arbitrary pattern $p(t, c) \in \mathcal{P}$ is prompted, for example, *It is [MASK] that computer science paper t belongs to the category c.* p, in this case, is a function of t and c as they can be arbitrarily plugged into it. The *[MASK]* will be filled with one of $\{Yes, No\} = \mathcal{O}$. We compute a probability distributions of all words using all classes in $\mathcal{C}$. The class with the highest probability of *yes* is chosen as the correct class c^*.

Computing the most likely classes is done through a classic softmax operation, as following:

$$c^* = argmax_c \frac{exp(P(\text{Yes}|p(c,t))}{\sum_{o \in \mathcal{O}} exp(P(o|p(c,t)))},$$

where $P(\text{Yes}|p(c,t))$ is the probability of *Yes* being given the pattern p with the current class c and the current title t.

Importantly, these experiments are conducted in a few-shot setting. BERT is able to handle up to 512 input tokens. That means before prompting the actual sample for which we want to infer the correct label, we can show the model a selection of positive and negative samples. Therefore, I chose to prompt additional 15 samples in the same format as the to be inferred sample. With 50% probability a true sample from the current class is shown (thus, with the label *true* or *correct*) and with 50% probability a random negative sample (thus, with the label *false* or *incorrect*) is shown.

Prompt	**Yes-No**	**Acc.**
It is [MASK] that computer science paper t belongs to the category c	True/False	5%
It is [MASK] that t is about c	True/False	7%
It is [MASK] that the paper titled t is about c	True/False	6%
It is [MASK] that the paper titled t belongs to the class c	True/False	5%
It is [MASK] that the paper titled t is about c	Correct/Incorrect	7%

Table 3.5: The five prompts tested in the binary classification few-shot setting on the ArXiv computer science dataset. Percentages indicated on the right are the accuracy for the respective prompt.

The set $\mathcal{P}$ of patterns I choose to test can be viewed in Table 3.5. The respective correct percentages for the ArXiv document classification task is annotated with it. Importantly to not, we have 20 classes at hand which are evenly distributed. Thus, the random baseline happens to be 5%. As can be seen in the table, the results do not look very promising and slightly evolve around the random baseline. Clearly, out of the possibilities tested, more variations in experiments could have been deducted. However, at this point I decided not to further investigate this direction and moved on to the Exact Pattern Matching approach

(next section). This proved to be a bit more interesting, thus the remaining time of this work was put into this.

3.5.5 Defbert's Zero-Shot Abilities: Exact Pattern Matching

In this section the exact pattern matching approach is explained. The high-level idea is the following. In a zero-shot setting, the model is prompted with a pattern. In this case the pattern p is a function of t only. We do not need to plug in all 20 classes as in the case of the binary classification. Given the pattern p, out of the 20 classes in $\mathcal{C}$ we are going to compute the probability of each class c. The pattern, for example could, be *The scientific paper t belongs to the category [MASK].* This time, for the [MASK]-token, we are not interested in determining which of *true* or *false* is correct. We are going to look for the probabilities of each of the 20 classes to be replaced with the [MASK]-token.

L	Title	Tokens
1	Databases	['databases']
1	Robotics	['robotics']
2	Machine Learning	['machine', 'learning']
2	Information Retrieval	['information', 'retrieval']
2	Software Engineering	['software', 'engineering']
2	Computational Complexity	['computational', 'complexity']
2	Artificial Intelligence	['artificial', 'intelligence']
2	Computational Geometry	['computational', 'geometry']
3	Computation and Language	['computation', 'and', 'language']
3	Systems and Control	['systems', 'and', 'control']
3	Computers and Society	['computers', 'and', 'society']
4	Cryptography and Security	['crypt', '##ography', 'and', 'security']
4	Logic in Computer Science	['logic', 'in', 'computer', 'science']
4	Networking and Internet Architecture	['networking', 'and', 'internet', 'architecture']
4	Data Structures and Algorithms	['data', 'structures', 'and', 'algorithms']
4	Human-Computer Interaction	['human', '-', 'computer', 'interaction']
5	Computer Science and Game Theory	['computer', 'science', 'and', 'game', 'theory']
5	Computer Vision and Pattern Recognition	['computer', 'vision', 'and', 'pattern', 'recognition']
7	Distributed, parallel, and Cluster Computing	['distributed', ',', 'parallel', ',', 'and', 'cluster' 'computing']

Table 3.6: The 20 classes in the ArXiv computer science dataset ordered in amount of tokens per title.

Different Token Amounts

One major problem that comes with this is that the names of the classes have different lengths. Thus, the tokenizer splits them into various sizes. In Table 3.6 one can observe the respective classes, with their token lengths. The issue that comes with different class lengths is the following. If we regard the pattern *The scientific paper t belongs to the category [MASK].* it is straightforward to obtain the probability for *databases* and *robotics*. These are just ordinary probabilities which can be obtained from the logits of the last layer of the model. However, in this setting, trying to obtain the probabilities for any of the other classes is simply not possible as more than one [MASK]-token is required. The idea is therefore to compute the probabilities for each of the six buckets $1, 2, 3, 4, 5, 7$ with its respective amounts of masks to be predicted. For example, if we wanted to obtain the probabilities for the class *Computation and Language* we would have to prompt *The scientific paper t belongs to the category [MASK] [MASK] [MASK].*. Obviously, the more tokens have to be inferred the lower the probability of the respective class to be selected. Clearly, we have to account for that.

Length Normalization

A naive approach to overcoming lower probability for having to infer more tokens would be normalizing the probabilities by the amount of tokens to be inferred. As probabilities are extremely low, we operate in log-space. If we, e.g., obtain a log-score of -80 for a class with 4 tokens we would simply divide by 4 to trim to log-probabilities to a similar level across all buckets. This has two downfalls. **1.** it assumes a linear decrease in probability as a function of length. This assumption likely is invalid. The more tokens have to be inferred, and the less prior information is available, the harder it is to predict the correct word. Likely, the difficulty of predicting the right tokens with an increasing token amount grows superlinearly. **2.** When operating in log-space division becomes subtraction, however, for this approach we still divide. This is not based on a scientifically fundamental approach but solely on observations and intuition. It does work fairly well, however, is hard to motivate.

Calibration

Another way to account for token amount is called *calibration* [80]. In principle for each of the classes a calibration constant is computed. This is done by computing the probability of the class *without* prompting a paper title. I.e., we would simply compute the probability of class c_i using the pattern $p(\text{""})$. This has been shown to be an effective measure in normalizing out various forms of biases [80]. In table 3.7 one can observe the constants for each of the classes. The numerical trend of the constants with respect to the title length can be seen in the plot in Figure 3.2.

#	**Title**	**Constant**
1	Databases	-0.0169
1	Robotics	-0.0125
2	Machine Learning	-1.6171
2	Software Engineering	-2.7063
2	Programming Languages	-2.1636
2	Computational Complexity	-2.9482
2	Artificial Intelligence	-0.5915
2	Computational Geometry	-2.7429
2	Information Retrieval	-2.8526
3	Computation and Language	-12.1547
3	Computers and Society	-11.1534
3	Systems and Control	-9.9880
4	Networking and Internet Architecture	-11.3654
4	Cryptography and Security	-13.21944
4	Logic in Computer Science	-10.1076
4	Data Structures and Algorithms	-14.2773
4	Human-Computer Interaction	-8.6135
5	Computer Vision and Pattern Recognition	-17.4370
5	Computer Science and Game Theory	-19.0024
7	Distributed, Parallel, and Cluster Computing	-15.0701

Table 3.7: Calibration constants for all 20 classes ordered by token length (#)

Moreover, there is another small problem which has to be addressed. It can happen that Defbert learns new words which are subwords of larger words that BERT already knew. For example, regard the term *retrieval* which has an OTR in BERT. On the ArXiv computer science dataset Defbert will learn the word *trie*. Due to the inner workings of the tokenizer, when encountering *retrieval* the tokenizer will split up the word into *[re, trie, ##val]*. Obviously, if *retrieval* has an already known OTR it is the better choice. This can be inhibited as following. For each word, compare BERT's and Defbert's tokenization.

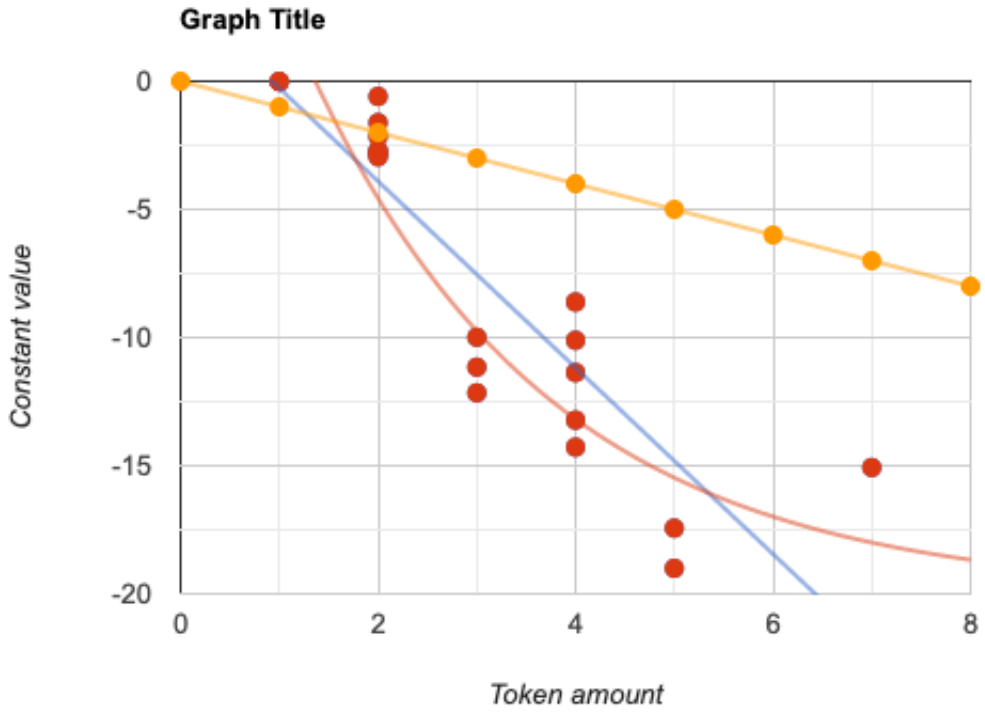

Figure 3.2: Linear trend (blue) and exponential trend (red) for the values of the calibration constants. Orange shows the identity function used to normalize by division.

If Defbert's tokenization is less than BERT's, fall back to BERT's tokenization. If we encounter a situation like this, it means that Defbert has created a subword representation which is part of a larger word. What originally seemed like a big problem turns out to not have a large effect on the test results. Likely, it is important to adapt the tokenizer accordingly. However, only 1% of all samples suffer from this issue.

Title	**DB**	**B**
The paper with the title t is about	20	10
The computer science paper with the title "t" belongs to the following category:	18	8
The computer science paper with the title t belongs to the following class:	16	10
The computer science paper with the title t is about	15	5
The paper with the title t belongs to the following category:	15	6
The paper with the title t belongs to the following class:	15	5
The paper t belongs to the following category:	15	7
The paper t belongs to the following class:	15	5

Table 3.8: Eight different prompts were tested in the exact pattern matching zero-shot setting on the ArXiv computer science dataset. **DB** stands for Defbert and **B** stands for BERT. The numbers in the respective columns show the accuracy in percent. In these experiments, length normalization is conducted.

The results for Defbert on the exact-pattern matching task with length normalization are shown in Table 3.8. We can see that Defbert performs better than BERT across all 8 patterns. It is hard to determine, however, why that is the case. As described in the previous section, normalizing log-space probabilities by dividing by token length is not scientifically foundational.

The results for Defbert tested in the exact-pattern-matching task using calibration has to be constructed a bit differently. Unfortunately, this makes it hard to compare to the length normalization approach. The reason for this is the following. In order to compute a calibration constant we need to compute the probability of an arbitrary class c without prompting the title. E.g., if the pattern was *The paper* t *is about [MASK] ... [MASK].* we would compute the calibration constants using the same pattern, without t: *The paper is about [MASK] ... [MASK].* This determines the general probability of class c without asking for a specific title.

This approach has the downside that patterns need to be constructed such that they make sense grammatically, even if the title is simply left out. As can be seen in Table 3.8 this is not the case for all of the patterns. For patterns 1 to 6, *with the title* is explicitly used. Thus, it is difficult to maintain this pattern only without using the title. In patterns 7 and 8 the definite article *the* is used referring to an explicit title. Unfortunately, only after having conducted a range of experiments this was noticed such that further adaptions were too late to conduct.

To overcome this - without the possibility of comparison - we construct a new prompt that is more flexible. Instead of utilizing the prompts shown in Table 3.8 we use the prompt *"The scientific paper belongs to the category [MASK]."*. This way, the prompt can be extended to *"The scientific paper ¡title¿ belongs to the category "* while retaining a correct grammatical structure.

When employing the above prompt for Defbert we attain 110 (9.2%) correctly classified documents and for BERT we obtain 106 (8.8%). The experiment was run for $\tau = 2000$ (see Section 3.4.2 for explanation) and conducted on the ArXiv computer science 1e+02 dataset. Unfortunately, differences are marginal thus at this point I decided not to further investigate this direction (also, due to time constraints).

3.6 Discussion

As can be seen in the results above, unfortunately, there are no clear signs of the approach increasing BERT's understanding of rare and domain specific words. Tough on WNLaMPro Defbert was on par with attentive mimicking [48], on the downstream task of the ArXiv document classification it was not possible to improve scores. There is a chance that Defbert embeddings indeed improve BERT's representation space. However, as mentioned before, most downstream tasks in NLP do not contain enough rare words to obtain a measure expressive enough.

The following few-shot and zero-shot settings, unfortunately, were not able to prove Defbert's capabilities either. There is a high chance that these settings where not tested extensively enough. While testing various prompts and configurations I noticed BERT's and also Defbert's extreme sensitivity to what is prompted. This is backed by current research in these settings [80].

Another limitation of the current study is the limited number of downstream tasks that were used to evaluate Defbert's capabilities. While the ArXiv document classification task is an important benchmark in the NLP community, it is important to note that other tasks and datasets may yield different results. Therefore, it is crucial to conduct further evaluations of Defbert on a variety of downstream tasks to fully understand its capabilities and limitations.

In Section 4.2 I broadly discuss measures that possibly went wrong when designing and conducting the experiments and point to further ideas.

4 Conclusion

4.1 Summary

In this thesis we employed a holistic view onto how simple definitions from Wiktionary or WordNet can be used to infer one-token-representations of arbitrary words. I motivated the topic, described the aim of the work and point out all methods deployed in Section 1. Fundamentals needed to understand the following sections were laid in Section 2.

The core of the the thesis is presented in Section 3. I explained in detail the data used to train the model and discuss its respective variations. Issues and downfalls are thoroughly discusses and accounted for. The approach tested is sensitive to a range of design choices which are gone over in the subsequent part of the chapter. The *pattern* used well as the exact settings to deploy training are presented in every detail.

The main part of the thesis is Section 3.5, evaluation of previously discusses methods. We test Defbert and its derivative hierarchical defbert in a wide range of settings. The focus of these methods, however, is retaining the pre-trained structure of the model. I.e., we never fine-tune the Defbert (or BERT) model such that the representation state is maintained. Thus, we primarily test the model on tasks involving the general PLM's capacity. This includes testing on WNLaMPro and the ArXiv document classification dataset. Moreover, we extend experiments to few-shot and zero-shot training. Unfortunately, no passable results indicating the usefulness of the approach were found. Importantly to note, no results were generated it is not clear whether the approach indeed does not work or whether experiments were not thorough enough. There is a high chance, especially in the zero-shot and few-shot settings, experiments enhanced by methods such as PET or MEAL yield better results [49, 24].

4.2 Outlook

There are a series of possible future steps to take to test this approach and possibly improve BERT's rare word understanding.

Definition Selection

Similar to Attentive Mimicking there's high chance that pre-selecting the right definitions may affect the effectiveness of the approach [48]. As shown in Section 3.2 quality of definitions differ significantly. Also, prevalent being that many deviations inherit extremely fine-grained senses which likely do not serve as good informants in many situations.

Few-Shot Learning

It has been shown that PLMs are extremely sensible to their prompts in few-shot settings. Possibly, the configurations tried in this thesis were not sufficient. There is a chance that other carefully engineered prompts work better than what has been tested. For example, I only tested a mix of 50% positive and 50% negative samples. But none of the experiments try only positive samples (which we have a sufficient amount for, usually negative samples are used to augment training data). Moreover, it has been shown that the order of shown samples play a significant role [80]. In addition to that, likely, there are samples which are simply more informative than others. Conducting more experiments in the few-shot setting may be promising but due to time constraints could not be evaluated in full depth.

Especially methods such as PET or MEAL possibly could have improved the setting and make few-shot inference a lot more stable. What could be observed is that prompts used turned out to be very inconsistent. It was hard to deduce why certain prompots worked better than others.

Zero-Shot Learning

Similar to the few-shot setting, testing in this setting can be extended further. It ended up not being clear which types of prompts have BERT outperform Defbert and vice versa. Other than that, one would have to investigate whether zero-shot learning in general is the tool to test BERT's abilities. Potentially, the language model is simply not large enough to conduct a task as demanding as this one.

Domain Specific Tests

Moreover, in this thesis I did not conduct tests on domain specific tasks. I claimed that it can be difficult to test a PLM's capabilities as in many NLP tasks only a limited subset of samples employs rare or domain specific words. We did evaluate the model on the ArXiv document classification task. But experiments could have been extended to other domains such as biology, finance or law.

Extended Experiments on ArXiv dataset

The zero-shot and few-shot approaches were solely conducted on the computer science part of the ArXiv dataset. Possibly, experiments could have been extended to the math and physics domain as well to obtain more reliable results.

Defbert's Implementation

Having a look at Defbert's actual implementation, it is unclear whether there is room for improvement, if the approach works and if it is only a matter of empirically proving this. When training Defbert and choosing the tokenizer's OTRs as targets this is a hard constraint which cannot be circumvented. It was not possible to come up with a way to augment the sample size (and likely there is none). Especially, what could be further investigated is the pattern used to train defbert.

More Models

In this work, experiments where only conducted on BERT. Possibly employing and testing the methods on RoBERTa or other deep PLMs such as T0 and ALBERT would have validated or falsified prior hypotheses [44, 26].

Bibliography

[1] Aguilar, Gustavo, et al. "A multi-task approach for named entity recognition in social media data." arXiv preprint arXiv:1906.04135 (2019).

[2] Araci, Dogu. "Finbert: Financial sentiment analysis with pre-trained language models." arXiv preprint arXiv:1908.10063 (2019).

[3] Ba, Jimmy, Volodymyr Mnih, and Koray Kavukcuoglu. "Multiple object recognition with visual attention." arXiv preprint arXiv:1412.7755 (2014).

[4] Baid, Palak, Apoorva Gupta, and Neelam Chaplot. "Sentiment analysis of movie reviews using machine learning techniques." International Journal of Computer Applications 179.7 (2017): 45-49.

[5] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).

[6] Bhardwaj, Rishabh, Navonil Majumder, and Soujanya Poria. "Investigating gender bias in BERT." Cognitive Computation 13.4 (2021): 1008-1018.

[7] Bisk, Yonatan, et al. "Piqa: Reasoning about physical commonsense in natural language." Proceedings of the AAAI conference on artificial intelligence. Vol. 34. No. 05. 2020.

[8] Bojanowski, Piotr, et al. "Enriching word vectors with subword information." Transactions of the association for computational linguistics 5 (2017): 135-146.

[9] Brown, Tom, et al. "Language models are few-shot learners." Advances in neural information processing systems 33 (2020): 1877-1901.

[10] Chalkidis, Ilias, et al. "LEGAL-BERT: The muppets straight out of law school." arXiv preprint arXiv:2010.02559 (2020).

[11] Choi, Hyunjin, et al. "Evaluation of BERT and ALBERT sentence embedding performance on downstream NLP tasks." 2020 25th International conference on pattern recognition (ICPR). IEEE, 2021.

[12] Chowdhery, Aakanksha, et al. "Palm: Scaling language modeling with pathways." arXiv preprint arXiv:2204.02311 (2022).

[13] Clement, Colin B., et al. "On the Use of ArXiv as a Dataset." arXiv preprint arXiv:1905.00075 (2019).

[14] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[15] Gu, Yu, et al. "Domain-specific language model pretraining for biomedical natural language processing." ACM Transactions on Computing for Healthcare (HEALTH) 3.1 (2021): 1-23.

[16] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[17] Henrich, Verena, and Erhard Hinrichs. "A comparative evaluation of word sense disambiguation algorithms for German." Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12). 2012.

[18] Herbelot, Aurélie, and Marco Baroni. "High-risk learning: acquiring new word vectors from tiny data." arXiv preprint arXiv:1707.06556 (2017).

[19] Hong, Jin-Hyuk, and Sung-Bae Cho. "A probabilistic multi-class strategy of one-vs.-rest support vector machines for cancer classification." Neurocomputing 71.16-18 (2008): 3275-3281.

[20] Izsak, Peter, Moshe Berchansky, and Omer Levy. "How to train bert with an academic budget." arXiv preprint arXiv:2104.07705 (2021).

[21] Jaech, Aaron, and Mari Ostendorf. "Personalized language model for query auto-completion." arXiv preprint arXiv:1804.09661 (2018).

[22] Khodak, Mikhail, et al. "A la carte embedding: Cheap but effective induction of semantic feature vectors." arXiv preprint arXiv:1805.05388 (2018).

[23] Kim, Yoon, et al. "Structured attention networks." arXiv preprint arXiv:1702.00887 (2017).

[24] Köksal, Abdullatif, Timo Schick, and Hinrich Schütze. "MEAL: Stable and Active Learning for Few-Shot Prompting." arXiv preprint arXiv:2211.08358 (2022).

[25] Kuchaiev, Oleksii, and Boris Ginsburg. "Factorization tricks for LSTM networks." arXiv preprint arXiv:1703.10722 (2017).

[26] Lan, Zhenzhong, et al. "Albert: A lite bert for self-supervised learning of language representations." arXiv preprint arXiv:1909.11942 (2019).

[27] Lee, Jinhyuk, et al. "BioBERT: a pre-trained biomedical language representation model for biomedical text mining." Bioinformatics 36.4 (2020): 1234-1240.

[28] Limsopatham, Nut. "Effectively leveraging bert for legal document classification." Proceedings of the Natural Legal Language Processing Workshop 2021. 2021.

[29] Lin, Tao, et al. "Extrapolation for large-batch training in deep learning." International Conference on Machine Learning. PMLR, 2020.

[30] Ling, Wang, et al. "Not all contexts are created equal: Better word representations with variable attention." Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. 2015.

[31] Lipka, Leonhard. An outline of English lexicology: lexical structure, word semantics, and word-formation. Vol. 3. Walter de Gruyter, 2010.

[32] Liu, Zhuang, et al. "Finbert: A pre-trained financial language representation model for financial text mining." Proceedings of the Twenty-Ninth International Conference on International Joint Conferences on Artificial Intelligence. 2021.

[33] Loshchilov, Ilya, and Frank Hutter. "Decoupled weight decay regularization." arXiv preprint arXiv:1711.05101 (2017).

[34] Luong, Minh-Thang, Richard Socher, and Christopher D. Manning. "Better word representations with recursive neural networks for morphology." Proceedings of the seventeenth conference on computational natural language learning. 2013.

[35] Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:1508.04025 (2015).

[36] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

[37] Mnih, Volodymyr, Nicolas Heess, and Alex Graves. "Recurrent models of visual attention." Advances in neural information processing systems 27 (2014).

[38] Moon, Sangwhan, and Naoaki Okazaki. "Effects and mitigation of out-of-vocabulary in universal language models." Journal of Information Processing 29 (2021): 490-503.

[39] Parikh, Ankur P., et al. "A decomposable attention model for natural language inference." arXiv preprint arXiv:1606.01933 (2016).

[40] Pelicon, Andraž, et al. "Zero-shot learning for cross-lingual news sentiment classification." Applied Sciences 10.17 (2020): 5993.

[41] Peters, M. E., et al. "Deep contextualized word representations. arXiv 2018." arXiv preprint arXiv:1802.05365 12 (1802).

[42] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[43] Pinter, Yuval, Robert Guthrie, and Jacob Eisenstein. "Mimicking word embeddings using subword rnns." arXiv preprint arXiv:1707.06961 (2017).

[44] Raffel, Colin, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." The Journal of Machine Learning Research 21.1 (2020): 5485-5551.

[45] Rajani, Nazneen Fatema, et al. "Explain yourself! leveraging language models for commonsense reasoning." arXiv preprint arXiv:1906.02361 (2019).

[46] Rajpurkar, Pranav, et al. "Squad: 100,000+ questions for machine comprehension of text." arXiv preprint arXiv:1606.05250 (2016).

[47] Salle, Alexandre, and Aline Villavicencio. "Incorporating subword information into matrix factorization word embeddings." arXiv preprint arXiv:1805.03710 (2018).

[48] Schick, Timo, and Hinrich Schütze. "Rare words: A major problem for contextualized embeddings and how to fix it by attentive mimicking." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. No. 05. 2020.

[49] Schick, Timo, and Hinrich Schütze. "It's not just size that matters: Small language models are also few-shot learners." arXiv preprint arXiv:2009.07118 (2020).

[50] Schick, Timo, and Hinrich Schütze. "BERTRAM: Improved word embeddings have big impact on contextualized model performance." arXiv preprint arXiv:1910.07181 (2019).

[51] Schick, Timo, and Hinrich Schütze. "Learning semantic representations for novel words: Leveraging both form and context." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 33. No. 01. 2019.

[52] Schick, Timo, and Hinrich Schütze. "Attentive mimicking: Better word embeddings by attending to informative contexts." arXiv preprint arXiv:1904.01617 (2019).

[53] Schick, Timo, Sahana Udupa, and Hinrich Schütze. "Self-diagnosis and self-debiasing: A proposal for reducing corpus-based bias in nlp." Transactions of the Association for Computational Linguistics 9 (2021): 1408-1424.

[54] Senel, Lütfi Kerem, Timo Schick, and Hinrich Schütze. "CoDA21: Evaluating Language Understanding Capabilities of NLP Models With Context-Definition Alignment." arXiv preprint arXiv:2203.06228 (2022).

[55] Sennrich, Rico, Barry Haddow, and Alexandra Birch. "Neural machine translation of rare words with subword units." arXiv preprint arXiv:1508.07909 (2015).

[56] Shi, Yue, et al. "Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering." Proceedings of the sixth ACM conference on Recommender systems. 2012.

[57] Sousa, Matheus Gomes, et al. "BERT for stock market sentiment analysis." 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI). IEEE, 2019.

[58] Souza, Fábio, Rodrigo Nogueira, and Roberto Lotufo. "Portuguese named entity recognition using BERT-CRF." arXiv preprint arXiv:1909.10649 (2019).

[59] Sun, Chi, et al. "How to fine-tune bert for text classification?." China national conference on Chinese computational linguistics. Springer, Cham, 2019.

[60] Sun, Chi, Luyao Huang, and Xipeng Qiu. "Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence." arXiv preprint arXiv:1903.09588 (2019).

[61] Sushil, Madhumita, Simon Suster, and Walter Daelemans. "Are we there yet? Exploring clinical domain knowledge of BERT models." Proceedings of the 20th Workshop on Biomedical Language Processing. 2021.

[62] Tang, Charlie, Nitish Srivastava, and Russ R. Salakhutdinov. "Learning generative models with visual attention." Advances in Neural Information Processing Systems 27 (2014).

[63] Triantafillou, Eleni, et al. "Learning a universal template for few-shot dataset generalization." International Conference on Machine Learning. PMLR, 2021.

[64] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[65] Wang, Alex, et al. "GLUE: A multi-task benchmark and analysis platform for natural language understanding." arXiv preprint arXiv:1804.07461 (2018).

[66] Wang, Yaqing, et al. "Generalizing from a few examples: A survey on few-shot learning." ACM computing surveys (csur) 53.3 (2020): 1-34.

[67] Wang, Yuxuan, et al. "Cross-lingual BERT transformation for zero-shot dependency parsing." arXiv preprint arXiv:1909.06775 (2019).

[68] Webersinke, Nicolas, et al. "Climatebert: A pretrained language model for climate-related text." arXiv preprint arXiv:2110.12010 (2021).

[69] Wei, Jason, et al. "Finetuned language models are zero-shot learners." arXiv preprint arXiv:2109.01652 (2021).

[70] Wieting, John, et al. "Charagram: Embedding words and sentences via character n-grams." arXiv preprint arXiv:1607.02789 (2016).

[71] Wu, Qiyu, et al. "Taking notes on the fly helps BERT pre-training." arXiv preprint arXiv:2008.01466 (2020).

[72] Wu, Ting-Wei, Ruolin Su, and Biing Juang. "A label-aware BERT attention network for zero-shot multi-intent detection in spoken language understanding." Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. 2021.

[73] Wu, Yang, et al. "Optimizing mean reciprocal rank for person re-identification." 2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS). IEEE, 2011.

[74] Wu, Yonghui, et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv preprint arXiv:1609.08144 (2016).

[75] Xu, Kelvin, et al. "Show, attend and tell: Neural image caption generation with visual attention." International conference on machine learning. PMLR, 2015.

[76] Yang, Yi, Mark Christopher Siy Uy, and Allen Huang. "Finbert: A pretrained language model for financial communications." arXiv preprint arXiv:2006.08097 (2020).

[77] Yang, Zhilin, et al. "Xlnet: Generalized autoregressive pretraining for language understanding." Advances in neural information processing systems 32 (2019).

[78] Yao, Liang, et al. "Traditional Chinese medicine clinical records classification with BERT and domain specific corpora." Journal of the American Medical Informatics Association 26.12 (2019): 1632-1636.

[79] Zhang, Ruixue, et al. "Rapid adaptation of bert for information extraction on domain-specific business documents." arXiv preprint arXiv:2002.01861 (2020).

[80] Zhao, Zihao, et al. "Calibrate before use: Improving few-shot performance of language models." International Conference on Machine Learning. PMLR, 2021.

[81] Zheng, Zhe, et al. "Pretrained domain-specific language model for natural language processing tasks in the AEC domain." Computers in Industry 142 (2022): 103733.