Pre-training large NLP-models by utilizing low-resource NLP-pipelines

van Cauter, Zeno

Award date:
2021

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science

Pre-training large NLP-models by utilizing low-resource NLP-pipelines

Master Thesis

ing. Zeno van Cauter (0935906)

Supervisor:
dr. habil. Cassio de Campos

Version 1.0

Eindhoven, September 2021

Abstract

The dominating approach to Natural Language Processing problems is by pre-training models on large amounts of unsupervised text and fine-tuning them for use on downstream tasks. Over the years, this amount of unsupervised text has grown almost exponentially to hundreds of billions of words. It is presumed that the pre-training through Random Masking and the large amount of pre-training data has lead to word co-occurrence becoming the main source of performance for these models. It is also presumed that the pre-training data is not used efficiently nor to its full potential.

Therefore this Thesis set out to introduces 4 new concepts to counter this problem. The first is *Strategized Masking*, which masks tokens based on Part-Of-Speech tags and lemmatization. The second is a Polynomial Time Approximation Scheme for so-called *Corpus reduction*, which reduces the total number of tokens in a corpus while minimizing the number of unique tokens lost. The third, *Chunking*, and fourth, *Split-training*, are concepts introduced to deal with sequences in the pre-training data, which are too long for the model to handle. They explore methods to reduce the token loss during pre-processing, as every token is of much higher value when subjected to *Corpus Reduction*.

All approaches include the use of spaCy's low-resource NLP-pipelines to put more supervision on the pre-training data, without the need for annotations.

It is discovered that *Strategized Masking* provides a significant improvement for pre-training loss and therefore filling masks. It also outperforms Random Masking significantly when compared on the GLUE-benchmark. For *Corpus reduction*, a 939M token dataset is reduced to a 28M, 10M, 1M and 100K version. The performance on downstream tasks show that there is indeed proof of a trade-off between the total number of tokens and token variety. While *Chunking* and *Split-training* reduce the numbers of tokens lost, they both significantly hurt performance. This is presumably due to an introduce a tokenization bias.

The results suggest that there is a clear advantage to using low-resource NLP-pipelines for the pre-training of large general-purpose NLP-models.

Foreword

This Thesis kicked off in February 2021, which was during the second COVID-19 pandemic lockdown in The Netherlands. The 3 quartiles before it started I had just finished a workload of 60 ECTS. The effects of the lockdown and previous work had a severe impact on my personal mindset that accompanied the start of this project.

At the start, I had several discussions with Cassio and with his repeated reassurances and comforting words, this project finally started to take off. The sometimes lengthy discussions that we had lead to this self-formed Thesis topic that I thoroughly enjoyed researching, regardless of all the roadblocks that were passed during it. Because of that, I cannot thank him enough for his patience, guidance and involvement as a supervisor.

While the start of this project was mostly done working from home, work really started progressing once people pushed me to come to the university library for self-study. Things such as small talk or getting coffee at the Pattern room made me realize how much I had missed the university during the lockdowns. The table we occupied was filled with (mostly) fellow Data Science students every day and it became a much nicer experience because of that.

I would also like to thank my girlfriend, Amber, for her love, support and understanding during the ups and downs that I experienced throughout this project.

Finally, I would like to thank my parents and sister. They have watched me claw through a mathematics degree at Fontys, which was followed by an unsuccessful pre-master and successful bachelor's degree at the TU/e. This Thesis and consequent master's degree will be final conclusion to the 10-year road that brought me here. During all these years they have kept supporting and rooting for me and I will be forever grateful for that.

Contents

Chapter 1

Introduction

In the last decade Natural Language Processing (NLP) models have moved from being mediocre sparse-matrix based classifiers [7] to all-encompassing language models which achieve state-of-the-art results on a large set of NLP-tasks [34][35], matching or even surpassing human performance. The current mainstream approach is to *pre-train* a model using a large amount of unsupervised textual data, followed by *fine-tuning* the model with a small amount of supervised data to make it task-specific.

The pre-training part consists of utilizing large amounts of textual data and training it on unsupervised objectives such as Masked Learning or Next Sentence Prediction [5]. For example: BERT [5] is pre-trained on roughly 3.3B words [5], the T5 [22] was pre-trained using 34B tokens and one of the current largest models, GPT-3 [2], is pre-trained on ∼400B tokens. This development is fairly logical, given that the increased availability of GPU/TPU-accelerated networks allowed for pre-training larger and larger models. This development in hardware combined with the increasing amount of publicly available textual data pushed the boundaries of NLP-models forward. This meant that progress of NLP research can be largely explained due to the use of bigger models and more data.

Examples of models that improved the pre-training of BERT are RoBERTa [13] or StructBERT [36], which both used more training data, were trained longer or used additional training objectives. However, neither of these models (both considered state-of-the-art performance) improved performance by an amount that warrants using even more lifetimes of data, nor did it improve language understanding of the models.

BERTology [23] has become the collective name for studies which try to probe NLP-models and explain the relation why the models work so well. A summary [23] suggests that BERT does not actually form a generic idea of named entities, despite scoring high on Named Entity Recognition tasks [1]. Other research attributes the state-of-the-art performance of BERT largely due to its massive increase in size [23] and what it actually learns is often by simple exploitation of distributional cues in the data [18]. In fact, research has shown that pre-training RoBERTa on sentences which have its words shuffled around randomly, will still achieves high accuracy on variety of tasks [27].

In conjunction with these probing results, a new movement has started against the computational expansion of NLP-models. Both the pre-training and use of large-scale NLP-models produce significant carbon emissions [28], lead to models being overparameterized [23] and lack the computational efficiency of its original inspiration: the human brain [25]. Originally, this comparison between the computational efficiency was mainly looked at from an energy consumption perspective [25], but the same argument can be made from the perspective of learning efficiency. Assuming that a person would read 30k words every day for 80 years, they would reach a lifetime total of

876M words. This means that BERT is trained on 4 lifetimes of text, the T5 uses 34 lifetimes and GPT-3 exceeding 400 lifetimes.

Based on this, there is an opportunity to take the critical look at the trend of ever increasing amount of pre-training data and how efficiently it is learned from.

1.1 Open challenges

The question on how to pre-train NLP-models using orders of magnitude less data is still unresolved. This Thesis will present 3 identifiable challenges, which are perceived to be useful in tackle the general one.

The first 2 challenges are based on the distinct difference between how humans are learned a language versus how computational models are perceived to learn. The last challenge is a result of how language models are practically implemented and relates to the second one.

The first challenge is how NLP-models are asked to learn from the pre-training data. The scale at which BERT is pre-trained and the methods to do so lead to word co-occurrence becoming the main source of performance [27]. This Thesis conjectures that the Masked Learning objective makes models guess blanks that may not be valuable to learning. The combination of the current method of Masked Learning and oversized datasets would cause and explain performance of pure word co-occurrence from [27]. One would rather have the model learn linguistic features in a targeted manner, rather than have the model form patterns fully unsupervised.

The second part is about the intelligent ways to select a subset of data for pre-training when the corpus is very big. It is unlikely that the entirety of a very large dataset is really necessary, given how redundant textual data is. The Brown Corpus (one of the first publicly available corpora), consisted of 1M words with half of them being the same 135 words [6]. A 2006 analysis of Project Gutenberg showed that 12% of the words present are accounted for with the words "the", "of" and "and"[1]. This means that there is a challenge to ensure "variety" in the text that its fed. Because large corpora are very likely to already have some form of this variety present, there is a challenge to distil it into a smaller corpus (preferably which could be read in a human lifetime) without losing that textual variation.

Third and last, is the one of efficient use of raw text. In the second challenge, every token of a reduced corpus is of much higher importance. Therefore, it becomes important to look at how the raw text data is turned into a suitable model input, and how tokens are lost during this process. From the source code of BERT [5], it becomes apparent that sequences are truncated to make them fit into the model. This becomes a problem when this may hurt linguistic diversity.

Resolving these 3 challenges should give a more efficient foundation for the circumstances under which NLP-models are commonly pre-trained.

1.2 Research Questions

So far, this chapter has covered the (possibly unnecessary) growth of datasets used to pre-train NLP-models. It also poses three challenges that can be used to tackle this issue. With these challenges in mind, this Thesis will attempt to cover three research questions to tackle these challenges:

RQ1: *What are methods that allow for NLP-models to be pre-trained in a targeted manner?*

[1] Retrieved August 31st 2021 from https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/PG/2006/04/1-10000

RQ2: *How can a large corpus which spans multiple human lifetimes of text be distilled into a smaller one without losing linguistic diversity?*

RQ3: *What are strategies that can minimize the number of lost tokens when dealing with sequences that exceed the maximum length of the model input?*

1.3 Related work

The original idea of pre-training in BERT was two-fold: obtain a large amount of unannotated text and have objectives for the model to learn from this text [5]. These objectives were Masked Learning and Next Sentence Prediction (NSP).

The approach of Masked Learning is to randomly mask a part of the sentence and have the model to guess the word. For example, if the training data contains the sentence "Today is a nice day for a walk", the model receives "Today is a nice [MASK] for a walk" and has to guess the original word on place with the [MASK] from some a-priori defined vocabulary $\mathcal{V}$.

Next Sentence Prediction (NSP) means that the model learns from long pieces of text (often originating from documents) and is paired with either the actual successive sentence in the text or a random sentence from the training data. For example, the sentence "Anne found a hole in her sock." can be combined with "So she went to the store to buy a new pair." or "The plane took off on its planned time.". The model then has to guess whether the sentence pair is composed of two successive sentences.

StructBERT [36] improved upon the pre-training methods by generalizing NSP to also include the previous sentence and by introducing a new training objective which consisted of shuffling around trigrams of unmasked words.

RoBERTa [13] was a replication study which investigated certain design choices made in [5]. More importantly, they found that leaving out the NSP-objective matches or slightly improved performance compared to the original approach in [5]. In addition, [13] trained using dynamic masking which means that the masks of sentences are different across epochs.

ELECTRA [3] went for a different approach in terms of Masked Learning. Instead of replacing the words by a [MASK] token, it uses another language model to generate suitable and hard to distinguish alternative tokens. Sequentially, a discriminator has to guess whether a word is original or replaced. The resulting advantage is that the model has to learn from a discriminative task between the original and a challenging alternative. This approach significantly increases the speed (in terms of FLOPS) at which the model learns from the training data.

In short, follow-up research has lead to extensions of the traditional training objectives or inspired other methods entirely. In fact, ELECTRA is one of the latest successes in reducing the required number of FLOPs needed to converge to a new state-of-the-art performance. However, ELECTRA still used 3.3B tokens during its pre-training process. In addition, there is little to no control or insight into what ELECTRA deems as "suitable alternatives" that would be hard for a model to distinguish.

Pre-training with different sizes of pre-training data has become an integral part of many papers [37] [32] [21] [10] [15]. The scale of data used for pre-training varies (sometimes millions, sometimes billions of tokens), but their conclusions form a common thread. A larger corpus helps, but with diminishing returns and more data is needed for subsequent improvements. It is likely that other forms of knowledge are the major drivers of recent improvements in language understanding among large pre-trained models [39]. Yet none of these approaches include a different method than random selection from a much larger corpus.

1.4 Approach

In accordance to the research questions formulated in Chapter 1, this Thesis dedicates 3 chapters to solving each research question separately.

For RQ1, Chapter 2 will introduce a form of non-randomized Masked Learning. This method is hypothesized to learn better than Random Masked Learning in multiple specific linguistic directions.

For RQ2, Chapter 3 introduces and demonstrates a method for reducing a corpus while maintaining token variety using a Polynomial Time Approximation Scheme (PTAS). Furthermore, it will cover how this PTAS can be extended to preserve other linguistic variations rather than only token representation.

For RQ3, Chapter 4 will present a new method of preserving tokens from sequences that exceed the maximum model length.

There is one silver lining in how all these 3 chapters are approached, which is the use of existing low-resource task-specific NLP pipelines (such as Part-Of-Speech taggers). All of these NLP pipelines are pre-trained using far fewer resources than the larger Transformer-based models. In a sense, these pipelines will be used to set the stage for the eventual training of such much larger NLP-models.

The approach of using other pre-trained NLP-models in a semi-supervised vision is not entirely new. [8] proposed a way of distilling knowledge from multiple models into a single one as an ensemble method. An adaptation of this approach is a Student-Teacher architecture [30] which involves training a student by using the output probability distribution from a teacher model. However, such methods rely on a larger model feeding the softmax probabilities to a smaller one. Instead, this Thesis will use small-scale NLP pipelines to select and optimize which and how the textual data is used for pre-training of a larger one.

After the explanation of solutions to the research questions chapter 5 explains which experiments will be performed and why. Specifically, it will introduce how the strategies presented in Chapter 3-5 are measured on successfulness. It also contains the details involved in the pre-training and fine-tuning of models.

The results from these experiments will be covered in chapter 6, with an extensive discussion on both the results and the used methodology.

Finally, this Thesis will present several new open challenges for future work in chapter 7.

The source code used during this Thesis is available on GitHub[2].

[2] github.com/zeno17/LessIsMore

Chapter 2

Strategized Masking

2.1 History of pre-training

For a long time, core NLP techniques were dominated by machine-learning approaches that used linear models such as support vector machines or logistic regression, trained over very high dimensional yet very sparse feature vectors [7].

The introduction of neural networks and subsequent word embeddings [16] provided new ways of looking at how specific tasks in NLP could be tackled. In addition, the non-linearity of the network, as well as the ability to easily integrate pre-trained word embeddings, often lead to superior classification accuracy [7]. The pre-training of these word embeddings involved having the model run through 1.6 billion words to learn a vector-representation for words in a vocabulary.

During this time, researchers were contending to find how the neural network architecture could be used for better and better performances on tasks. For example, tasks such as sentiment classification, paraphrase identification or part-of-speech tagging found themselves having new and improved performance by neural networks [7][1].

Another development that was becoming more popular in this period was that of transfer-learning. Re-collecting the needed training data and rebuild the models for every task is expensive, meaning the transfer of knowledge between task domains could be desirable [19]. This means that it could be useful to for example, use a model trained on sentiment classification and transfer its knowledge for paraphrase detection. In the field of NLP it became apparent that this practice could be utilized to its best potential by pre-training the model on a general purpose linguistic objective, and then transferring that knowledge to downstream tasks. In some sense, this was already happening as word embeddings were pre-trained on objectives such as Next Word Prediction or n-grams in order to embed linguistic features into the word embeddings. To this day, virtually all state-of-the-art models still use embeddings (word-embeddings, positional embeddings, etc) as the primary input.

CoVe [14] was one the first to apply transfer learning at a large scale, by pre-training an NLP model through a large amount of unsupervised text (7M sentences) and fine-tuning afterwards. CoVe pre-trained its model by use of an encoder-decoder structure and Machine Translation data. By given the model a sentence in a source language and asking it to predict the translation in a target language, the encoder would learn more about the linguistic structure of the source language and the decoder would learn more about the target language. By decoupling the encoder, it could be fine-tuned to fulfill downstream tasks in the source language.

[1] The citation provides a non-comprehensive list of 32 different tasks. Due to the fact that the amount of different NLP-tasks has grown in the years since then, only 3 from the original list are provided here for the sake of space

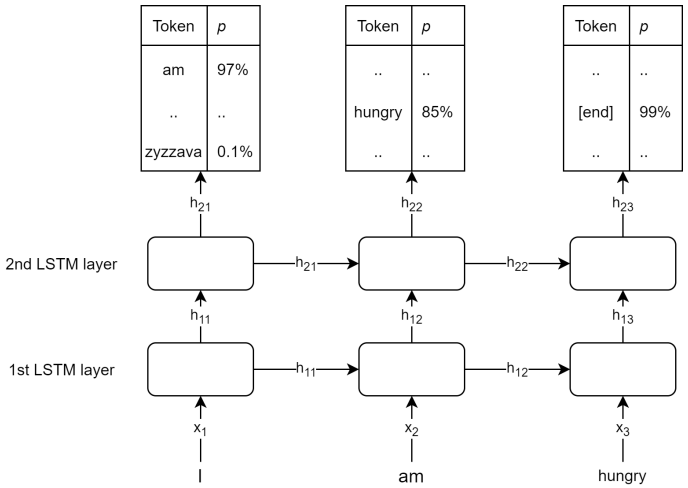


Figure 2.1: Example LSTM for Next Word Prediction.

ULMFiT [9] was the first model which no longer changed architectures (such as decoupling an encoder-decoder) when switching to specific tasks. Instead, it only changed the last layer to allow for task specific classifications.

All the architectures and approaches mentioned so far have something in common, which is the unidirectionality in which pre-training learned. Figure 2.1 shows an example for the Next Word Prediction objective with a simplified view of the Long Short-Term Memory (LSTM) architecture. The predictions are recurrent and the model has to predict the next word basic on everything it has seen so far.

The model can also placed in the backward direction by flipping the horizontal arrows in Figure 2.1. Combining both directions gives the Bidirectional LSTM (BiLSTM) architecture used in CoVe/ULMFiT. The problem here is that the model is now looking both forwards and backwards in separate constructs rather than all at once. This is where the Transformer architecture [33] comes in.

2.2 Masking

The Transformer architecture introduced another encoder-decoder structure with a new internal neural mechanism called attention. For BERT, the model only needed the encoder part, which allows the model to look at both directions at once. The problem with this bidirectionality is that objectives such as Next Word Prediction becomes useless for general purpose pre-training, because the model can just "peek" across its layer for what the answer should be.

This lead to the introduction of Masked Learning, where the model is given the entire sequence with certain parts blanked out (inspired by the Cloze task [29]). The model then has to learn to fill the blanks with the original words, as an objective for general purpose pre-training. An illustration is given in Figure 2.2. The end-result is that the model can now utilize both the forward and backward perspective, but cannot peek into either side to cheat the pre-training objective.

BERT applied random masking, meaning that the masks were applied to tokens at random. Because the Transformer architecture has remained the dominant approach throughout developing research, masking has also remained as the dominant pre-training objective. Common approaches are Random WordPiece Masking (Figure 2.3a) [38] or Random Whole Word Masking (Figure 2.3b).

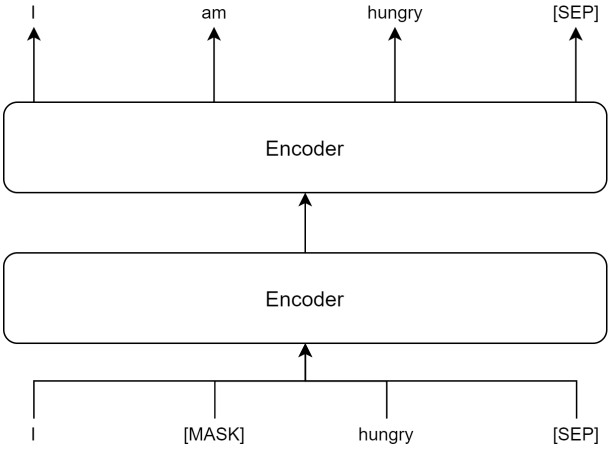
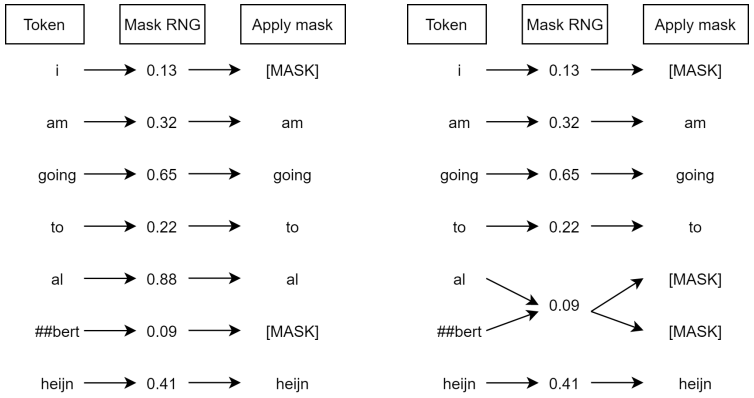


Figure 2.2: Masking example for BERT.



(a) Random masking (b) Random Whole Word Masking

Figure 2.3: (a) Implementation for Random Masking. Each token gets rolled a number from a uniform [0,1] distribution. Every token with a number below the threshold (in this case, 0.15) gets masked. (b) Implementation of Random Masking. Here, the number is rolled for every word. Then, words are masked in its entirety or not at all.

For both Random WordPiece Masking and Random Whole Word Masking, the mask remained the same throughout different pre-training steps. Later on, the concept of "Dynamic Masking" was introduced, where the masks are shuffled every training step. This means that a sequence can have different masks based on which batch it is in.

Even ELECTRA, which provides hard to distinguish replacements instead of the [MASK] token still utilizes the random selection of tokens. However, this Thesis conjectures that pre-training would benefit from a more targeted learning for pre-training, which can be explained by an analogue to how humans are learned reading comprehension skills.

When children are presented a Cloze task, they are not presented with randomly blanked out parts of the sentence. For example, a child can be asked to fill in the blank in "An elephant is _____, but a mouse is _____". Basic reading comprehension skills allows people to make the connection that the words being sought are adjectives that are descriptive of the nouns: "big" and "small".

Based on this analogy, this Thesis conjectures that model pre-training would benefit from a similar

form where the masks are based on linguistic features rather than a random selection.

2.3 Learning linguistic features

One of the core aspects of textual data is that text can be high-dimensional regardless whether it is supervised or unsupervised. This means that for any sentence, there are a multitude of perspectives to absorb information. Consider the following sentence:

> "Jimmy went to the store at 7 o'clock to buy vegetables before the store closes."

This sentence entails several deducible facts: (1) Someone is going to do something; (2) It is about someone going somewhere at a certain time; (3) The whereabouts of the described person is unknown, but it is known where the person is going; (4) The person is going to do something at the location where he is going towards. This is also why BERT works so much better than previous architectures, every word has a certain bidirectional connection to other words in the sentence (so-called "dependency parsing").

In addition to these factual deductions, the sentence contains hidden information about linguistic features: (1) expression of place and time in the same sentence, which has an order of first place, and then time; (2) Every word actually has a class prescribed to it (verb/noun, so called Part-Of-Speech tags) which appear in an order that make grammatical sense. In this case, a noun ("Jimmy") followed by a verb ("went").

Obtaining supervised classifications (annotations) for these linguistic features can be cumbersome, e.g. obtaining Part-of-Speech tags, sentiment (positive/negative/neutral) or a parse tree for a single sentence is expensive. It is therefore logical that pre-training objectives such as Next Word Prediction and subsequently Masked Learning became the mainstream approach, it does not require the annotation of billions of words or sentences.

The question is then, how can pre-training be done in a targeted manner which does not require the annotation of the pre-training data?

2.4 Strategized Masking

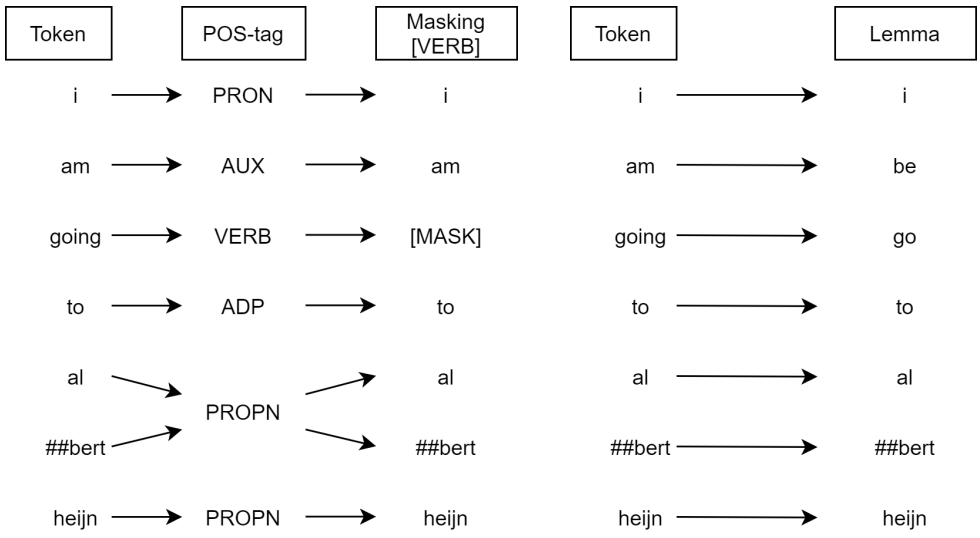
As specified earlier, this Thesis conjecture is that this method of masking is one of the causes of memorization by token co-occurrence as described in [27]. Therefore, this Thesis will present a new method: "Strategized Masking".

With Strategized Masking, the sequence is masked based on actual linguistic features rather than masking tokens at random. This includes the use of spaCy[2] [17] and utilizing its NLP-pipelines to classify the words in a sequence and mask based on these tags. Figure 2.4a is an example of how spaCy's POS-tagger can be used to mask verbs in a sequence. Figure 2.4b demonstrates how spaCy's lemmatizer can be used to create grammatically incorrect (but linguistically close) sentences that the model needs to correct. Finally, the Named-Entity-Recognition (NER) parser can be used to recognize concepts such as "time" which can be modified to be grammatically incorrect. An example of this is given in Figure 2.5.

Using this way of masking, the model has to guess masked tokens in a directed manner in a way that imitates grammatical structures[3]. It forces the model to learn about specific correct grammatical structures (e.g. "I am" and not "I be") or notations (e.g. "5 o'clock" and not "o'clock 5"). This should lead to a more white-box learning of supervised structures while maintaining the possibility to use unsupervised data.

[2] https://spacy.io/

[3] The grammar rules are inspired by www.learnenglishkids.britishcouncil.org/grammar-vocabulary



(a) POS-based (b) Lemmatization

Figure 2.4: (a) POS-based masking. Every word gets assigned a Part-Of-Speech tag and sequences can be masked accordingly, such as masking all verbs in a sequence. (b) Lemmatization.

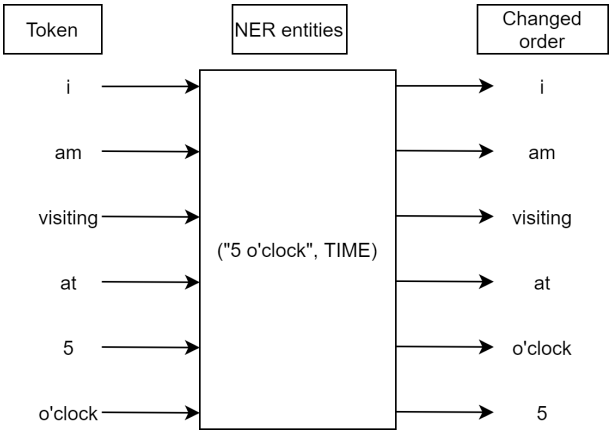


Figure 2.5: Switching the token order of specific parts that were recognized by the NER. In this example, notation of time.

2.5 Error-robustness of Strategized Masking

SpaCy allows us to get state-of-the-art classifications and parsing but like any prediction model ever, it is subject to classification errors[4]. Therefore, it should be shown or argued that Strategized Masking is robust to this. The main argument that will be provided here is that misclassifications in the components of Strategized Masking are related to strategies as in BERT [5] or StructBERT [36].

1. **Part-Of-Speech Tagging**: Misclassifications in the POS-tagger lead to some sentences having masks which are not all from the same actual class. This means that when a sequence which should have all and only verbs masked, has a non-verb masked as well. This outcome is not very different than the current mainstream approach where masking is done at random.
2. **Named-Entity-Recognition Parser**: Misrecognition will lead to having non-NER objects having swapped around tokens. This is similar to how the "Word Structural Objective" is used in [36], which was shown to be a viable training strategy.
3. **Lemmatization**: SpaCy lemmatizes by having a long look-up list of words with their lemmatized versions and having basic grammatical rules [5]. If a word is incorrectly lemmatized, it can be viewed as a random replacement comparable to [5].

In all 3 scenarios, the worst case scenario is a sentence mask which is not fully focused on 1 specific linguistic structure. Albeit not perfect, it should still result in a far more directed manner of learning than that of Random Masking.

2.6 Summary

This chapter showed how current mainstream approaches for masking work and introduced a non-randomized form of Masking which is conjectured to improve upon these mainstream approaches. It utilizes NLP-pipelines of the spaCy library in an error-robust way which targets specific linguistic features to be learned during pre-training.

The performance of this newly introduced masking method will be verified experimentally in chapter 5 and chapter 6.

[4] Accuracy can be found at https://github.com/explosion/spacy-models/releases/tag/en_core_web_sm-3.1.0

[5] www.github.com/explosion/spacy-lookups-data

Chapter 3

Downsizing a pre-training corpus

3.1 Redundancy in textual data

Pre-training of NLP-models is done using large collections of textual data. Examples of these datasets are: the BooksCorpus used in the original BERT paper [40] or the Colossal Clean Common Crawl (c4)[1] [21]. Larger datasets such as c4 contain a billion sequences, but comes at the cost of requiring an enormous amount of disk space: 6.83 terabytes[2]. The result of using this approach is that NLP-models are trained on corpora spanning multiple lifetimes of text.

A lot of this data is redundant. When c4 is cleaned as per methods described in T5 [21], it leaves 364M sequences at at 866 gigabytes. These method mostly revolve around resolving web-scraping artifacts or deduplicating long identical sequences. Some words are more frequent than others, and in the case of the training data it can be considered overly frequent depending on the source.

Because NLP-models are pre-trained using gradient descent, every batch costs one optimization step. This also means that every sequence which is included into our training data adds computational cost, because it needs to be engrained into the model. For most research the solution is to either dramatically increase the batch size, like RoBERTa [13] using a batch size of 8K compared to 64 in BERT [5]. However, this does not touch upon the core problem at hand, which is the redundancy present in a lot of text. This Thesis conjectures that this large data redundancy forces the model to focus on learning word co-occurrences for performance as detailed in [27].

Instead, pre-training should be much more enforcing of "learning efficiency" regarding linguistic diversity. The distribution of word frequency (how much a word occurs in overall language use) is well-established to be related to Zipf's Law [20]. Naturally, a model cannot be pre-trained on a corpus which would have a corrected distribution (e.g. removing overly frequent words to make it uniform), because it would yield a broken and unnatural language. Instead, the goal is to only allow overly frequent words into the training corpus when it paired with something that adds linguistic diversity.

This chapter will set out to do 2 things. First, it will present a proof of concept on how a large redundant corpus can be "reduced". The focus here lies primarily on rebuilding the corpus while only allowing overly frequent words which introduces other unseen words. The second part will

[1] A cleaned version based on the original data available at commoncrawl.org

[2] The original source (www.tensorflow.org/datasets/catalog/c4) gives a size in TiB (TebiByte) which is converted to metric units.

be to show how this proof of concept is both scalable to corpora much larger than the one used here, and how it can be generalized to include other more complex linguistic features.

3.2 Dataset and vocabulary

To demonstrate the concept of corpus-reduction, a large subset obtained from the corpus of Project Gutenberg[3] (PG) will be reduced into a smaller one. Only English books which yield uncorrupted[4] files will be considered. This leaves a base corpus of 12.640 books.

While the vocabulary of people is highly dynamic, NLP-models use static vocabularies for computational reasons. These vocabularies are artificially made to include the most frequent words and leave the rest to WordPiece tokenization [38]. Therefore, both the size and contents of the vocabulary are the result of a prior desired computational complexity rather than a linguistic feature.

The vocabulary mainly decides the effect of the corpus-reduction as well as the produced outcome. Given that the main focus in this Thesis is the concept of corpus reduction itself, it will not focus on the factors involved in vocabulary construction. For convenience, it will use the already "pre-trained" vocabulary of *bert-base-uncased*. The end of this chapter will reflect on how this vocabulary made from Wikipedia data introduces a selection bias when combined with the data of Project Gutenberg.

The dataset is then cleaned, which entails the stripping of tables, figures, PG disclaimers, transcriber notes, footnotes, title pages, etc. Splitting the data on delimiters[5] results in a collection bunch of lines/paragraphs per book.

While cleaning is an integral part of the data selection for pre-training, it is rarely discussed in detail. The mainstream course of action is to mention high-level details, such as: *"For Wikipedia we extract only the text passages and ignore lists, tables, and headers"* [5]. Sometimes nothing is mentioned at all, nor is the source code provided [13]. To break this trend, the source code used to clean Project Gutenberg is available on this projects' GitHub such that those interested can obtain the same cleaned raw text.

After cleaning, this corpus consists of ∼939M tokens in total and contains 27.833 unique tokens. The challenge now lies ahead to reduce this number of total used tokens without losing any of the unique tokens.

3.3 Corpus-reduction under a token-variety constraint

Based on what has been specified for use so far, it is time to reflect back upon the initial research question: how can a dataset with 939M tokens across 12.640 books be reduced into a smaller one?

The most basic solution to begin with is by making a new "bookshelf" and whenever a book comes across that would introduce new words, it is added to this new shelf. This means that, instead of looking at every sentence or sequence on its own, there is an aggregate level which forms the basis for what enriches the bookshelf and what does not. However, books can differ massively in length, text complexity, targeted audience, type of story and many other factors. But this perspective allows an entry point to demonstrate this concept in a computationally feasible way.

[3] www.gutenberg.org

[4] Using the www.github.com/c-w/gutenberg library for retrieval of the text files. Sometimes it could not retrieve files because they were corrupted.

[5] It actually checks for multiple forms of delimiter ("\n\n", "\r\n\r\n") because PG is not standardized.

3.4 Mathematical formulation

Now that the overall corpus is looked at on a book-level, the problem can be formulated mathematically. Let $\mathcal{B}$ be a set of available books in our dataset. For every book b_i, all text can be processed to count how often every token occurs, a so-called token frequency table. Using the token frequency table, a token presence table can be made to represent which tokens are present in every book b_i. Figure 3.1 displays this process.

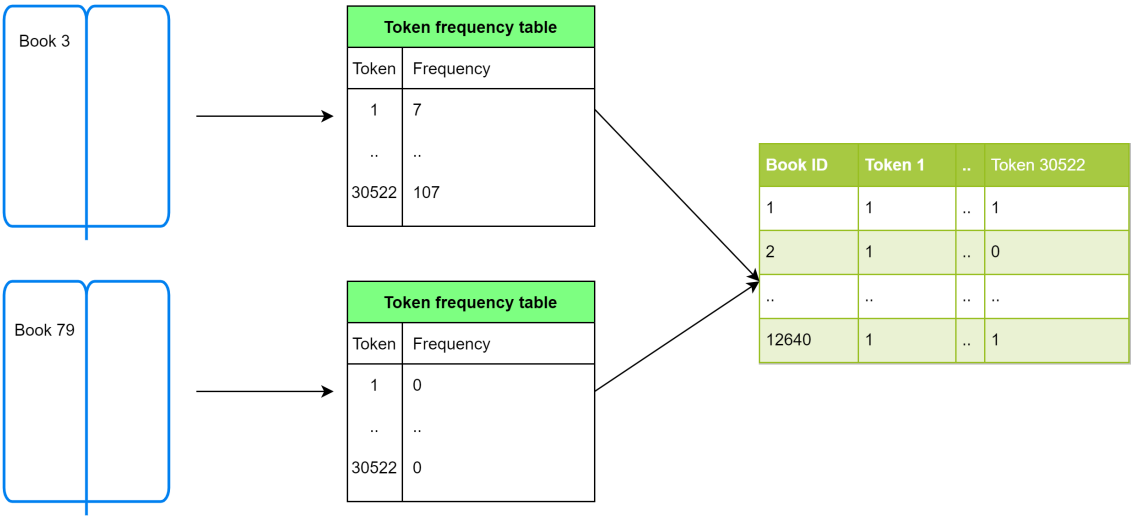


Figure 3.1: Processing of a book into a token frequency table, followed by the token presence table which shows books have which tokens present (represented in binary).

Each book b_i has a total number of tokens n_i (sum of the token frequency table) and a set of unique tokens $\mathcal{T}_i$ (all 1's in the token presence table). Let $\mathcal{T}_\mathcal{B}$ be all tokens which are available in the entire dataset. The desired result is a set of books $\mathcal{C} \subseteq \mathcal{B}$ which contains all tokens $\mathcal{T}_\mathcal{B}$, using the smallest amount of tokens (not the smallest amount of books). In terms of the original problem, how can the number of words in the corpus be reduced without losing any unique words present? This can be formulated as an optimization problem as given in Equation 3.1.

$$\begin{aligned} \min \quad & \sum_{c \in \mathcal{C}} n_c \\ \text{s.t.} \quad & \bigcup_{c \in \mathcal{C}} \mathcal{T}_c = \mathcal{T}_\mathcal{B} \end{aligned} \tag{3.1}$$

Given that the desired solution is a certain combination of books taken from the original 12.640 books, this problem is of combinatorial complexity. It is complicated even further by the fact that some tokens may appear in multiple or all books (e.g. frequent words such as "for", "and", "is" will appear in more books).

Along with this, every book in $\mathcal{C}$ will contribute a different amount of unique tokens depending on all other books in the composition. This means that to obtain the optimal solution, there is simply no other way than checking for every composition whether it has all tokens (the constraint in Equation 3.1) and finding which composition uses the least amount of tokens to do so. This means that the overall complexity is that of equation Equation 3.2.

$$\mathcal{O}\left(\frac{|\mathcal{B}|!}{(|\mathcal{B}| - |\mathcal{C}|)!}\right) \tag{3.2}$$

Based on this highly undesirable complexity, there are options that will make this problem easier to deal with. This first step is to relax the original problem in Equation 3.1 into something else, such that $|\mathcal{C}|$ will be small (and thus term in the denominator of Equation 3.2 will be large). The second step is to use approximations algorithms. For the sake of time and scope of this Thesis, both steps are explored and combined.

3.4.1 Relaxation

The formulation in Equation 3.1 has the constraint that all tokens need to be represented. This problem can relaxed by instead of requiring it to represent all tokens, have it try to maximize the amount of tokens it represents, using only a prior defined limit of tokens.

This means that the problem can be rewritten into a multi-objective optimization problem with the goals of a minimum amount of tokens used and a maximum amount of tokens represented, while staying under a certain amount of tokens. Defining $\mathcal{L}$ as the token threshold (the maximum number of tokens that can be used) leads to a double objective optimization problem as given in Equation 3.3.

$$\begin{aligned} \min \quad & \sum_{c \in \mathcal{C}} n_c, \qquad \max \quad |\bigcup_{c \in \mathcal{C}} \mathcal{T}_c| \\ \text{s.t.} \quad & \sum_{c \in \mathcal{C}} n_c \leq L \end{aligned} \tag{3.3}$$

While the new problem is now an optimization problem with 2 disjoint objectives (both a minimization and a maximization which are not nested), it has also reformulated the original problem into an extended version of the traditional Knapsack Problem[6], with weights n_i and with the value being the amount of unique tokens contributed by book c. The Knapsack Problem is known to be NP-complete, meaning that the corpus reduction problem will also be at least NP-complete as well. Nonetheless, there is now a threshold parameter $\mathcal{L}$ which can be used to steer for the size of $|\mathcal{C}|$. If the threshold $\mathcal{L}$ is small, it is expected to have less books in the solution, and therefore $\mathcal{C}$ will be smaller. The advantages are two-fold: (1) the problem is now related to a well-known optimization problem with related approximation algorithms and (2) it allows for making different datasets of at most size $\mathcal{L}$ and its different solutions $\mathcal{C}_\mathcal{L}$.

3.4.2 Approximation algorithm for the relaxed problem

The second step is to take the relaxed problem from Equation 3.3 and develop a Polynomial-time approximation scheme (PTAS). Based on the fact that the reformulated problem is now an extended version of the Knapsack Problem, it allows the design of an approximation scheme inspired from existing PTAS for that problem. This results in a greedy approximation approach as given in algorithm 1. Starting with an empty "Bookshelf", keep adding books which have the highest ratio of book tokens divided by the amount of new tokens it adds (based on an existing Knapsack PTAS by [4]). The algorithm halts when it either hits the threshold or can no longer introduce new unseen tokens. The latter case would give an approximate solution for the original formulation in Equation 3.1 without the relaxation.

[6] en.wikipedia.org/wiki/Knapsack_problem

Algorithm 1: Greedy-Ratio Bookshelf Builder

Data: A set of books $\mathcal{B}$, a threshold $\mathcal{L}$ and the unique set of available tokens $\mathcal{T}_\mathcal{B}$ available from $\mathcal{B}$

Result: A combination of books $\mathcal{C}$ which has less than $\mathcal{L}$ total tokens

1 $\mathcal{C} \leftarrow \{\}$ // Current combination of books
2 $\mathcal{T}_\mathcal{C} \leftarrow \{\}$ // Current unique tokens represented by combination $\mathcal{C}$
3 $n_\mathcal{C} \leftarrow 0$ // Number of total tokens in combination $\mathcal{C}$
4 $\mathcal{B} \leftarrow$ set of all available books
5 **while** $n_\mathcal{C} \leq \mathcal{L}$ *and* $\mathcal{T}_\mathcal{B} \setminus \mathcal{T}_\mathcal{C} \neq \emptyset$ **do**
6 **for** *book* $b_i \in \mathcal{B}$ **do**
7 $n_i \leftarrow$ total number of tokens in book b_i
8 $\mathcal{T}_i \leftarrow$ unique tokens in book b_i
9 $\Delta_i \leftarrow \mathcal{T}_i \setminus \mathcal{T}_\mathcal{C}$ // New unique tokens that adding book i would introduce to $\mathcal{T}_\mathcal{C}$
10 $r_i \leftarrow \frac{|\Delta_i|}{n_i}$ // Ratio between total tokens in the book and amount of newly introduced tokens
11
12 **end**
13 $b_{best} \leftarrow argmax(r_i : i \in 1, \ldots, |\mathcal{B}|)$ // Book with best ratio
14 $\mathcal{C} \leftarrow \mathcal{C} \cup b_{best}$
15 $\mathcal{T}_\mathcal{C} \leftarrow \mathcal{T}_\mathcal{C} \cup \Delta_{best}$
16 $n_\mathcal{C} += n_{best}$
17 **end**
18 return $\mathcal{C}$

3.5 Runtime complexity and space optimization

For optimization purposes, $\mathcal{T}_i$ and n_i (featuring in the token frequency table and token presence table in Figure 3.1) can be precomputed and stored to disk. This precomputing has complexity of linear time in the total number of tokens: $\mathcal{O}(\sum_i n_i)$ (all tokens in all books need to be processed once). Storing a single $\mathcal{T}_i$ requires $|\mathcal{V}|$ bits (a token is in $\mathcal{T}_i$ or its not and can therefore use a binary representation). This means that precomputation and storage requires $\mathcal{O}(|\mathcal{B}| * |\mathcal{V}|)$ space if stored in full. If the dataset consists of short texts with few different tokens, sparse storage is much more efficient and only requires $\mathcal{O}(\sum_i \mathcal{T}_i)$ bits. The advantage of this precomputation is that retrieving both $\mathcal{T}_i$ and n_i will now require constant time: $\mathcal{O}(1)$.

Deducing the full running time of the greedy approximation algorithm is brief. Line 5 opens a loop of $\mathcal{O}(|\mathcal{B}|)$, because at worst all books need to be added. Line 6 opens another loop of $\mathcal{O}(|\mathcal{B}|)$. Lines 7, 8 and 10 are all $\mathcal{O}(1)$ (as previously stated) inside these loops. Line 9 uses a Numpy implementation which is capped by the complexity of MergeSort[7] and hence is in all cases equal to:

$$\mathcal{O}((|\mathcal{T}_i| + |\mathcal{T}_\mathcal{C}|) * log_2(|\mathcal{T}_i| + |\mathcal{T}_\mathcal{C}|)) \tag{3.4}$$

Because $|\mathcal{T}_\mathcal{C}|$ tries to approximate the vocabulary $|\mathcal{V}|$ (it is trying to include as many different tokens as possible) and that in the worst case it will be $|\mathcal{T}_i| = |\mathcal{T}_\mathcal{C}| = |\mathcal{V}|$ and therefore the complexity of line 9 can be simplified to $\mathcal{O}(|\mathcal{V}| * log|\mathcal{V}|)$. Tracing this back through the loops means that the algorithm has a worst case running time of:

$$\mathcal{O}(\mathcal{V}| * log_2|\mathcal{V}| * |\mathcal{B}|^2) \tag{3.5}$$

[7] Based on the source code inspection.

However, this theoretical worst requires an almost artificially constructed case which is unlikely to occur using real textual data. The absolute worst case occurs when every book only contains 1 specific unique token (the frequency of this token in a single book is irrelevant because of the precomputation of $\mathcal{T}_i$), which would result in line 5 being executed $|\mathcal{B}|$ times (because every book needs to be added). In practice, the algorithm halts earlier after having processed $\mathcal{C}_\mathcal{L}$ books. Meaning that the practical case (not average case) is:

$$\mathcal{O}(\mathcal{V}| * log_2|\mathcal{V}| * |\mathcal{C}_\mathcal{L}| * |\mathcal{B}|) \tag{3.6}$$

The result is an approximation scheme which is the multiplicative of 3 factors: log-linear in the size of the vocabulary, linear in the size of the expected selected combinations and linear in the amount of books that it started with.

Having touched extensively upon the running time of the greedy approximation algorithm, an estimate for performance guarantees will not be covered in this Thesis.

3.6 Applying the algorithm

Using the corpus from Project Gutenberg, algorithm 1 is executed using the following different values for $\mathcal{L}$: 1e+5, 1e+6, 1e+7 and 1e+8. The results are given in Table 3.1.

$\mathcal{L}$	Total tokens	$\lvert\mathcal{C}_\mathcal{L}\rvert$	$\lvert\mathcal{T}_\mathcal{C}\rvert$
100K	99.974	51	13.040
1M	999.825	178	24.294
10M	9.977.907	656	27.607
100M	28.660.288	828	27.833
Full dataset	∼935.000.000	12.640	27.833

Table 3.1: Results from the Greedy-Ratio Shelf Builder algorithm. Full original dataset added for reference.

Using a subset of 656 books and a threshold of $\mathcal{L} = 10$M tokens, the subset $\mathcal{C}_{10M}$ already contains 99.2% of all tokens available in the corpus. Using only 828 books with a total of ∼28.6M, all tokens available in the corpus are represented, reducing the number of tokens by a factor of 32.6. These confirm the original suspicion that a lot of the training data is redundant (from a token perspective), and upholds the conjecture that pre-training may benefit from being more selective on the data used.

3.7 Scalability & Generalisation

So far, this chapter has introduced a greedy polynomial time approximation algorithm that allows for the reduction of a corpus where content is only added if it introduces new linguistic value. For computational reasons, it was decided to use the predetermined vocabulary of *bert-base-uncased* and using Project Gutenberg at a book-level. However, the proposed method for corpus reduction can be deployed at much larger corpora (albeit with some challenges), much larger vocabularies and a richer language representation that goes much further beyond than initially presented.

3.7.1 Dealing with larger corpora

Utilizing the data from Project Gutenberg was straightforward by using existing libraries and was computationally light. The dataset was effectively reduced from 12.640 books containing 935M tokens to 828 books containing 28.6M tokens without losing any token in the data in under a few

hours. While this worked fine for the current settings case, a larger dataset (billions of tokens) may run into issues from multiple perspectives.

The first would be the memory issue (and in very large cases maybe disk space as well) for the $\mathcal{O}(\sum_i \mathcal{T}_i)$ requirement for sparse storage. In the case that this could not be resolved by running it on a high-RAM machine, it could use a form of distributed computing (e.g. Spark/Hadoop MapReduce) which performs line 9 in parallel across multiple machines.

The second would be the case where $|\mathcal{B}|$ grows large. This would be the case if instead of running the algorithm on book-level, it would have run on paragraph level. Another case would be when using a large page-level dataset (such as English Wikipedia, which consists of over 6.3M pages[8]). Like earlier where the problem was changed to include a threshold $\mathcal{L}$, a new hyperparameter k can be introduced which can be used to augment line 13. Instead of picking the top-1 (book with the best ratio), the top-k best ratios are picked. This would reduce the original complexity from Equation 3.2 into:

$$\mathcal{O}(\mathcal{V}| * log_2|\mathcal{V}| * |\mathcal{C}_{\mathcal{L}}| * \frac{|\mathcal{B}|}{k}) \tag{3.7}$$

In section 3.6, Project Gutenberg was run on a book-level with $k = 1$. Which means that k was a factor of $\sim 0.0001 * |\mathcal{B}|$ and results with the same complexity. In the case where $|\mathcal{B}|$ is much larger (such as Wikipedia), a desired k can be selected with a trade-off between approximation accuracy and speed.

3.7.2 Extension to incorporate specific linguistic structures

Last, it is important to highlight that this solution for corpus reduction can be extended to enrich corpora with specific language structures using existing NLP-pipelines.

Words inside the vocabulary are atomic units while their meaning depends on the sequence containing them. The word "start" can represent a verb: "to start with something", or it can represent a place "He is at the start". While this computational challenge was mostly dealt with by the introduction of the Transformer-architecture [33], the training data would benefit from containing both distinct examples.

Using existing small-scale NLP-pipelines an extended *proto-vocabulary* $\mathcal{V}^*$ can be built, which not only consists of the words themselves but also on whether they represent different linguistic meaning (such as Part-Of-Speech tags). This allows for capturing different dynamics of language into a richer dataset, but can also be collapsed into the original vocabulary.

Again using the word "start" as an example, assume that the vocabulary $\mathcal{V}$ contains the word 'start' as a whole (no WordPiece). A POS-tagger finds that this word can have 2 different POS tags assigned to it, namely "VERB" or "NOUN" (assuming both are in the data). This means that both 'start-VERB' and 'start-NOUN' are added to the proto-vocabulary $\mathcal{V}^*$. Using the new proto-vocabulary in the algorithm, it will now value adding a new word-tag combination because it introduces new linguistic information.

With this extension, the corpus reduction will not only maintain token variety, but also polysemy. There is no doubt that there are further extensions possible which can give rise to pre-training data which is less redundant yet just as diverse.

3.8 Selection bias of the corpus

The selection of English books from Project Gutenberg was preprocessed using the pre-trained vocabulary $\mathcal{V}$ from *bert-base-uncased* which has a vocabulary size of $|\mathcal{V}| = 30.522$ tokens. Given that only 27.833 tokens were present in the entire dataset, it would seem that the selected corpus

[8]Retrieved 9th of June from https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia

would be missing representation of 2.689 tokens from the vocabulary. However, upon inspection it can be seen that 994 tokens are of the class '[unused]' which allow model-users to customize the model and 1.493 tokens are characters from alphabets of unseen languages (Greek/Hebrew/Arabic/Chinese/etc.) or WordPiece tokens such as "##a".

For the remaining 202 tokens it quickly becomes apparent that there is a clear case of selection bias, given that the entire corpus is sourced from Project Gutenberg while the vocabulary is based on Wikipedia. Examples of "more modern" tokens that are missing in Project Gutenberg are "lgbt", "mvp" or "xbox". There are also cases where the characters used on Wikipedia differ from the notation on PG, e.g.: 'km^2' instead of 'km2'.

Regardless, this is not a validity issue of this project or this Thesis. It is to an extent more the issue of taking a base corpus which does not have enough words represented from the overall language. This issue could be resolved by using the Colossal Clean Crawled Corpus [21] instead of the Project Gutenberg dataset, which was used to limit the scale and scope of this thesis.

3.9 Summary

This chapter introduced a PTAS for corpus reduction, designed to reduce redundancy in textual data while maintaining token variety. It included a complexity analysis and a small-scale demonstration which reduced the Project Gutenberg corpus by a factor of 32 without losing any unique tokens. Finally, it provided several future extensions that expand into expanded linguistic variation and scalability to much larger corpora.

The performance on the datasets produced by this newly introduced method will be verified experimentally in chapter 5 and chapter 6.

Chapter 4

Data loss during preprocessing

Chapter 3 demonstrated how a large corpus can be reduced to a smaller one while maintaining linguistic diversity. However, tokens can still be lost based on how raw text is transformed into suitable model input. Losing tokens through this process would be of much higher concern because when the corpus is reduced, every selected token is of much higher importance.

Therefore, this chapter will focus on techniques that can be used to both minimize the loss of tokens during preprocessing.

There are two phases which are of high influence on how the original raw data is actually used: (1) cleaning and (2) tokenization. The cleaning methodology has been explained in section 3.2, therefore this chapter will only detail the part of tokenization.

4.1 Tokenization

Tokenization at its core is the process of turning text into numeric inputs. Figure 4.1 demonstrates how every text is changed into a sequence of token IDs using a prior defined vocabulary. In addition, two other tokens are added: "[CLS]" and "[SEP]" which are artifact tokens from the implementation for BERT.

NLP-models have a maximum sequence length that it can handle as input, which is a hyper-parameter and can be changed based on computational complexity desires. While the example sequence in Figure 4.1 is fairly short, others are of much longer nature. The paragraphs retrieved from Project Gutenberg have large differences in length and complexity. Story-telling books which are dialog based can have very simple affirmations as a single paragraph:

> '*Yes.*'
>
> — "The Bad Man" by Charles Hanson Towne

It is also possible to have long introductory paragraphs with complex linguistic structures that refers within itself, paragraphs before it and paragraphs coming afterwards:

> "*I have so far treated blockade as the initial stage of a struggle for the command of the sea. That appears to me to be the logical order of treatment, because when two naval Powers go to war it is almost certain that the stronger of the two will at the outset attempt to blockade the naval forces of the other. The same thing is likely to happen even if the two are approximately equal in naval force, but in that case the blockade is not likely to be of long duration, because both sides will be eager to obtain a decision in the open. The command of the sea is a matter of such vital moment to both sides that each must needs seek to obtain it as soon and as completely as possible,*

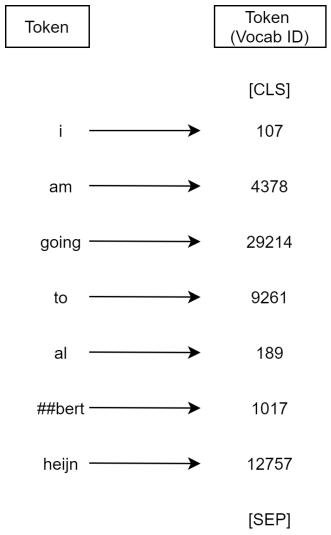


Figure 4.1: Example of tokenization. Token ID's are exemplary and not based on the action implementation of *bert-base-uncased*.

> *and the only certain way to obtain it is by the destruction of the armed forces of the enemy. The advantage of putting to sea first is in naval warfare the equivalent or counterpart of the advantage in land warfare of first crossing the enemy's frontier. If that advantage is pushed home and the enemy is still unready it must lead to a blockade. It is, moreover, quite possible that even if both belligerents are equally ready (I am here assuming them to be approximately equal in force) one or other, if not both, may think it better strategy to await developments before risking everything in an attempt to secure an immediate decision. In point of fact, the difference between this policy and the policy of a declared blockade is, as I am about to show, almost imperceptible, especially in modern conditions of naval warfare. It is therefore necessary to consider the subject of blockade more in detail. Other subjects closely associated with this will also have to be considered in some detail before we can grasp the full purport and extent of what is meant by the command of the sea.*"
>
> — "Naval Warfare" by James R. Thursfield

In BERT the basic approach was to combine short sequential sentences (such as the simple affirmation) into longer sequences. 90% of the time they aim for the maximum sequence length, 10% of the time they aim for a length somewhere between 2 and the maximum sequence length. The reason for this is according to the source code that: *"we *sometimes* want to use shorter sequences to minimize the mismatch between pre-training and fine-tuning"*, although this was never proven through its own or other research[1]. Nonetheless, it is assumed that it is important to have sequences of a different lengths during pre-training. This method results in two scenarios.

The first scenario is when sequences exceed the maximum length of the model input. The paragraph about naval warfare consists of 387 tokens and would be more than 3x too long for a 128-token length model. One of the larger paragraphs in the Project Gutenberg dataset is even 1.887 tokens long[2]. Truncating excessive tokens to make it fit a model with maximum sequence length 128 would mean removing 93.3% of the tokens. In addition, every truncation of tokens (small or large) means an increased likelihood that a unique token may be truncated from the dataset. That is a problem because after reducing a corpus, every token is of much greater import-

[1] All of this is based on source code inspection.
[2] "*Christian Mysticism*" by William Ralph Inge.

ance to the overall pre-training process. A way to induce no data loss would be to make a model that accepts an input much longer than any sequence in the pre-training data and leaving the positional embeddings remain untrained. Obviously, this is undesirable because it would leave a large part of the parameters untrained and make it immensely inefficient for computations.

The other scenario is where sequences are very short. In fact, the simple affirmation of "Yes." is only 4 tokens long: [SEP], "yes", ".", [CLS]. Most implementations for Transformer-based models cannot use a batch where the sequences are of a different length (including the one used here), which means that if a short sequence ends up in the same batch as a long one, the shorter one needs to be padded to match the longer ones length. If this sequence needs to padded to the maximum model input length (128 for our case), it would need to be padded with the "[PAD]" token 124 times. With a maximum sequence length of 128, 96.9% of the computing capacity would be wasted on "[PAD]" tokens.

Therefore the scope during this Thesis will be two-fold: (1) When sequences are too long, how can data-loss be minimized? and (2) how can the computational efficiency be improved for short sequences?

4.2 Minimize data-loss

The question at hand is how to deal with sequences which exceed the maximum sequence length of the model. Naturally, there are multiple ways to do this. A sequence could split into chunks which are exactly of length 128. This ensures there is no token loss, but may yield very broken sentences, especially when tokens that start with the continuation characters ("##ing") are at the beginning of the next sequence. While that can be resolved by removing this continuation characters from the start, it may still leave very broken sequences that start mid-sentence.

Instead, utilizing NLP-pipelines (such as spaCy's Sentence Segmentizer) a sequence could segment a paragraph into sentences which can then be reattached such that it only just exceeds the maximum sequence limit. This introduces some *data-loss* but decreases the likelihood that the model gets fed sequence which starts mid-sentence or with a continuation token. A visual demonstration can be found in Figure 4.2.

Using this method saves much more tokens than simply truncating sequences which are too long. While it does not eliminate the loss of tokens entirely, it prevents the formation of broken sequences that will mismatch with sequences given in downstream tasks.

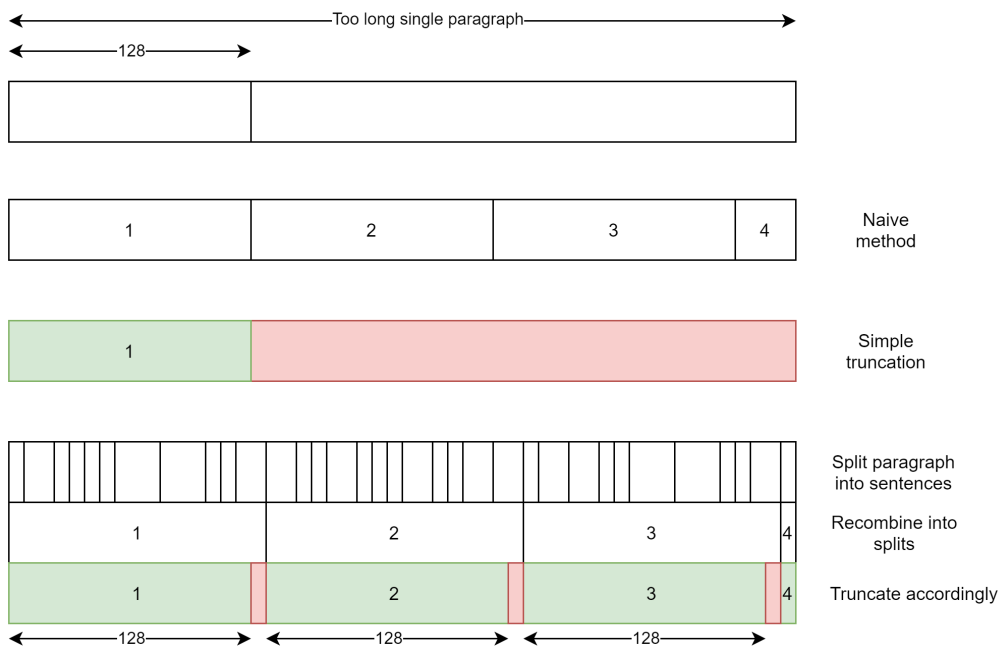


Figure 4.2: The problem of dealing with a paragraph which exceeds the maximum model input length. 3 methods are displays. (1) Naive (making splits of exactly 128 tokens and a remainder); (2) Truncation (3) Chunking (using a sentence segmentizer to determine sentence boundaries).

4.3 Maximize sequence efficiency

It was shown earlier that short sequences can lead to inefficient training when padded. For BERT, the solution was to glue sequential sentences/paragraphs together with a small probability of creating short sequences. However, this gluing process can once again lead to sequences which are too long, putting the problem of the previous paragraph back on the table.

The biggest problem at hand is that a model cannot handle sequences which are of different length, demonstrated in Figure 4.3. This is due to the technical limitations of GPUs which, while greatly improving the execution of gradient descent, cannot deal with non-rectangular inputs. This means a forward and backward pass is only possible if the original input is rectangular, meaning sequences must be of the same length.

Frankly, there is a fundamental perspective missing when dealing with these short sequences, which is the simplicity often present in short sentences. A new language is often earned by learning short sentences such as "How are you?" or "Where is the supermarket?" instead of long paragraphs about naval warfare. From this perspective, there seems to actually be value in pre-training specifically with shorter sequences and scaling up.

BERT does actually pre-train with different sequence lengths as well. The first 90% of its steps on length 128 and last 10% of its steps on length 512. However, these sequences may compose much more complex language than shorter sequences. It is therefore conjectured that pre-training with different maximum sequence lengths is beneficial as a form of "warm-up" for much longer sequences.

This leads to the introduction of the concept of split-training. For this project, which uses a maximum sequence length of 128, the dataset is split into 3 different splits:

Split 1: Sequences with a length up to 8 tokens
Split 2: Sequences with a length between 8 and 32 tokens

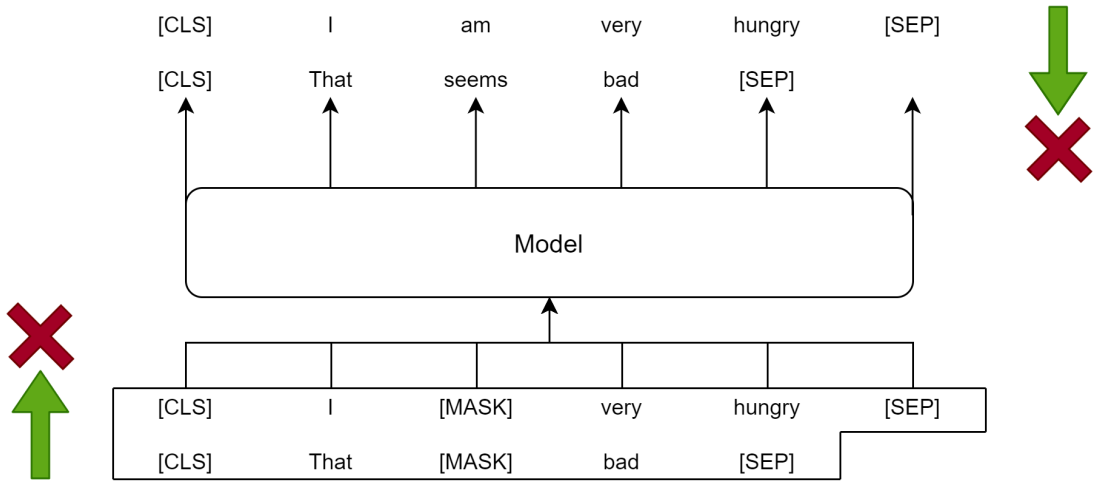


Figure 4.3: Example of trying to pass sequences with different lengths through the model. It can be seen that the inputs at the bottom are of non-rectangular shape and therefore both a forward-pass and backward-pass is impossible on a GPU.

Split 3: Sequences with a length between 32 and 128 tokens.

This is followed by padding sequences up to the maximum length corresponding to the split they are in. Sequences 4 tokens long are in split 8 and are padded up to length 8, etc.

This fits into two views on this process. The first is that of computational efficiency per sequence fed to the model: if sequences require less padding, the efficiency per "relatively empty" sequence goes up. The second is that short sequences will now fulfill a purpose of warming-up the model to longer sequences.

4.4 Summary

This chapter introduced two new concepts that can be used during tokenization and pre-training: chunking and split-training.

Under the constraint of a reduced corpus as in chapter 3, every token is of much higher importance. This means that too long sequences are chunked into shorter ones using an NLP-pipeline.

Sequences which are short compared to the maximum sequence length wastes computing power during pre-training, as it processing the padding token which is not useful. Therefore split-training is introduced which reduces the wasted efficiency. It also prevents from sequences having to be glued together which will introduce unnecessary new data-loss.

The performance of these newly introduced tokenization methods will be verified experimentally in chapter 5 and chapter 6.

Chapter 5

Experiments

This Thesis set out to answer 3 research questions related to using a smaller pre-training dataset: (1) What are methods that allow for NLP-models to be pre-trained in a targeted manner? (2) How can a large corpus which spans multiple human lifetimes be distilled into a smaller one without losing linguistic diversity? (3) What are strategies that can minimize the number of lost tokens when dealing with sequences that exceed the maximum length of the model input?

This was followed by 4 new concepts designed these questions: Strategized Masking, a PTAS for corpus reduction, chunking and split-training.

While each of these strategies has been demonstrated to work as a concept, the question remains whether these concepts will improve, match or hurt performance on downstream tasks. To verify this, experiments will be performed check the impact of these concepts on the GLUE-benchmark.

This chapter will discuss the experiments that will be performed, the choices made regarding both pre-training and fine-tuning, and it will detail a technical description such that these results can be reproduced.

5.1 Hardware

During this Thesis, extensive use was made of two different computation environments. One was the High-Performance-Cluster (HPC) provided by the Technical University of Eindhoven, the other was Google Colab[1].

For the HPC, this meant access to a Xeon Platinum 8260, Xeon Gold 6134 and 2x Tesla V100 16 GB. For Colab, it was dependent on what was available, which usually amounted to whether a Tesla-P100 or a Tesla-V100 (the former being the most common) was available[2]. The reason for using both the HPC and Google Colab stems mostly from the fact that the HPC was sometimes unreliable, unavailable and it did not respect the queuing order of jobs. For example, when process A was already running and process B got queued, it would always try to run process B in parallel. In the case of trying to pre-training a model with the largest possible batch size, this would often result in a memory error and abortion of the job. In an attempt to circumvent this, process B would be queued in an exclusive mode (where it would wait until all other running processes would be finished). However, processes from other users would skip ahead and be run in parallel. This resulted in a scenario where it would be unclear when (if ever), the GPUs on the HPC would be fully available. Given the troubling circumstances regarding long duration pre-training sessions on the HPC, it led to the additional use of Google Colab.

[1] The pro version was used.
[2] The experiments were performed before the introduction of Colab Pro$^+$.

For Colab, most practical issues came from the 24-hour session limit, which required writing several checks and checkpoints to ensure that everything was trained and fine-tuned accordingly. Practically, this meant using simple pickle (.pkl) files to track which models had been pre-trained, which models were not pre-trained yet and which models completed which tasks for fine-tuning.

Overall, a large amount of time/effort/coding time had to be dedicated specifically to be able to run experiments around the technical limits of the HPC and Colab. It also cost considerable time and effort to move data/models from the HPC to Google Colab after the HPC turned out to be inadequate.

5.2 Preprocessing data

For the preprocessing, raw .txt files from Project Gutenberg were cached (stored on local disk) and could be reused without straining the PG-servers/mirrors every time code was executed. The first step was to run algorithm 1 to retrieve $\mathcal{C}_{100K}$, $\mathcal{C}_{1M}$, $\mathcal{C}_{10M}$ and $\mathcal{C}_{100M}$ as described in chapter 3. Due to the unreliability of the TU/e HPC, this project lacked the hardware availability to preprocess and pre-train using the original unreduced corpus (of ∼939M tokens). It was not possible to perform this process with Google Colab either.

Next, the raw text data was processed as described in chapter 4, which returned appropriate sequences for pre-training.

5.3 Types of experiments

Based on the 4 introduced concepts, 4 experiments will be conducted.

The first will be a comparison between the original Random Whole Word Masking and the Strategized Masking. In addition, it will also include 2 reduced versions of the Strategized Masking technique, one with only the lemmatization and the other with only POS-based masking.

The second experiment will be a comparison between the different sizes of pre-training data. This means pre-training 1 model for every datasize: $\mathcal{C}_{100K}$, $\mathcal{C}_{1M}$, $\mathcal{C}_{10M}$ and $\mathcal{C}_{100M}$.

The third experiment will entail a comparison of tokenization methods. It will test whether there is a difference in performance between chunking and truncation. This requires the data to be preprocessed twice: once using truncation and once using the chunking method.

The fourth and final experiment is an investigation on split-training. It will test whether it is useful at all to use it, but also test the allocation of the computational budget across the different split lengths.

5.4 Technical details

Every experiment is run on two different model sizes: BERT$_{base}$ and BERT$_{tiny}$ [31]. This is done to test the experiments at different scales. Due to a limited computational budget, 100k steps are used for BERT$_{base}$ while BERT$_{tiny}$ gets 200k steps. For comparison, both BERT and RoBERTa's largest models used 1M steps during pre-training [5] [13].

Models are pre-trained using $\mathcal{C}_{100M}$ by default, unless specified otherwise such as experiment 2. In addition, all experiments will make use of split-training unless specified otherwise. This means that the model is first trained with sequences up to length 8 (split 1), followed by sequences between length 8 and 32 (split 2) and finally using sequences between 32 and 128 (split 3).

For BERT$_{base}$, the default step distribution is 30k/30k/40k. This means that split 1 gets 30k steps, split 2 gets 30k steps and split 3 gets 40k steps. In similar fashion, BERT$_{tiny}$ has a step distribution of 60k/60k/80k.

For computational complexity reasons, a maximum sequence length of 128 is used instead of the 512 from BERT. This reduction in maximum sequence length is in line with research such as ELECTRA [3]. Next Sentence Prediction is not used as a pre-training objective as it was shown to mirror or even hurt performance [13].

For pre-training, all default parameters of Trainer class were used except for 3: (1) a warm-up ratio of 0.1 (2) a learning rate of 1e−4 (3) weight decay of 0.01. This was done so that the hyperparameters matched those of the original BERT paper [5].

BERT_{tiny} is pre-trained with a base batch size of 128. This means that for split-training split 1 uses a batch size 2048, split-2 uses batch size 1024, and split 3 use the base batch size. For BERT_{base}, pre-training is done using a base batch size of 32. his means that for split-training split 1 uses a batch size 512, split-2 uses batch size 128, and split 3 use the base batch size. Increasing the batch size when sequences are shorter was done in order to maximize the usage of the GPU.

All both pre-training and fine-tuning are done on English datasets, as only the English books from Project Gutenberg were selected. However, all methods described in this Thesis are applicable to other languages aswell, as long as those languages also have the required NLP pipelines.

5.5 Evaluation

5.5.1 GLUE

For the experiments, the models performance will be measured on the GLUE-benchmark [35]. By using the GLUE benchmark, it allows the models to be evaluated across a diverse set of 9 Natural Language Understanding (NLU) tasks [35]. The performance of the models on this benchmark will allow as a measure for how much of linguistic structure is ingrained by the model. These tasks include: linguistic acceptability (CoLA), sentiment analysis (SST-2), paraphrasing (MRPC), paraphrasing for questions (QQP), textual similarity (STS-B), textual entailment (MNLI, MNLI-mm and RTE), question-answer entailment (QNLI), adverserial reading comprehension (WNLI) and language diagnostics (AX). Table 5.1 displays the corresponding metadata of these tasks.

Task	Train size	Validation size	Test size
CoLA	8.551	1.043	1.063
SST-2	67.350	872	1.821
MRPC	3.668	408	1.725
QQP	363.846	40.430	390.965
STS-B	5.749	1.500	1.379
MNLI	392.702	9.815	9.796
MNLI-mm	392.702	9.832	9.847
QNLI	104.743	5.463	5.463
RTE	2.490	277	3.000
WNLI	635	71	146
AX	392.702	9.815	1.104

Table 5.1: Metadata for the GLUE tasks. MNLI and MNLI-mm use the same training data but have different validation and test sets. For AX, the model finetuned on MNLI is used and only has a seperate test set.

5.5.2 Loss-benchmark

While training loss is not a widely used tool for performance measurement, the loss progression in both Figure 6.1 and Figure 6.2 sparked interest for measuring the loss over the entire pre-training

data.

For this part, all sequences from split 3 (length between 32 and 128) from $\mathcal{C}_{100M}$ were selected and masked in two different ways: Random Masking and Strategized Masking. For every model the loss was measured over this dataset and compared to how much epochs it had seen of split 3 used for pre-training.

This benchmark did not require any fine-tuning as it only requires the fully pre-trained models with mask-filling capabilities.

5.6 Fine-tuning technical details

For fine-tuning, a hyperparameter search is performed ($n = 10$, similar to ELECTRA [3]) with the following variations: (1) a learning rate between 1e−5 and 1e−4; (2) the number of epochs between 1 and 5; (3) batch sizes of 16, 32 or 64; (4) different seeds for batch selection.

For computational complexity reasons, the training data is reduced during the parameter search. The tasks 'MNLI', 'MNLI-mm' and 'QNLI' have their training data sharded to 10% of the original. For 'SST-2' the training data is sharded to 20% of the original. For 'QQP', both the training data and the validation data are sharded to 10% of the original.

Using the optimal hyperparameters from the 10 searches, the final fine-tuned model is trained with the full training dataset instead of a sharded one. This model is then used to make predictions for the test set and these predictions are submitted to the GLUE-benchmark[3].

The test data is not directly accessible and requires submitting a .zip with predictions to the GLUE-benchmark website. This prevents "cheating" by submitting predictions which are already optimized on the test-data. Every model is submitted only once and the results are definitive.

5.7 Summary

This chaptered detailed which experiments will be performed to test the new concepts introduced in chapter 2, chapter 3 and chapter 4. It detailed the technical details of hardware, pre-training and fine-tuning. It also provided specifics of how the evaluation on the GLUE-benchmark and loss-benchmark will be completed. The results can be found in chapter 6.

[3] https://gluebenchmark.com/

Chapter 6

Results

This chapter contains the results for the pre-trained and subsequently finetuned models. Results are split into different sections based on the comparison. Every table contains the results for both BERT_{base} and BERT_{tiny}. The best results will be given in bold. The metric is provided in the table with "ACC" denoting accuracy, "MCC" as an abbreviation for Matthews Correlation Coefficient and "F1" denoting the F1-score. Some tasks will have both the F1-score and accuracy denoted.

Unfortunately valid predictions could not be obtained for the STS-B benchmark. Therefore those results are omitted. Some models may appear multiple times across different tables, which was done to provide an easy comparison to the reader.

For some models, tasks still failed to train even with the hyperparameter optimization. Because some of the benchmarks contain class imbalances, some models will achieve higher accuracies by only predicting a single class (majority classifier). Despite a model failing to provide any meaningful predictions for a task, it can obtain a high "performance" due to this imbalance. In this case the results will be striked out. Following both BERT and ELECTRA, the results for WNLI are omitted due all models being incapable to beat this majority classifier [3].

6.1 Pre-training

Figure 6.1 and Figure 6.2 demonstrates the loss progression during the pre-training of the models. Because split-training is used, the loss for sequences with a length of up to 8 tokens reduces very rapidly. Once it encounters sequences between length 8 and 32, it has untrained positional embeddings and possibly new tokens as well. This causes the loss to spike and resettle again. This will repeat a second time for the third split.

What is more interesting is how the pre-training loss of Random Masking compares to those using Strategized Masking. It is important to be reminded here that these models use the same loss function (Categorical Cross-Entropy loss) and batch size. It is therefore also extremely surprising that the loss of Random Masking stays as high as it does during pre-training.

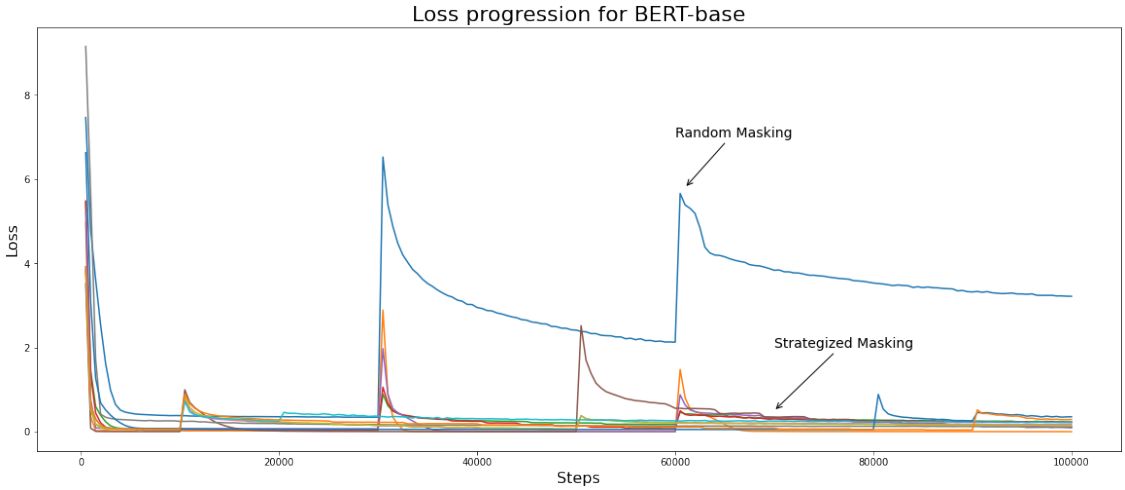


Figure 6.1: Loss during the pre-training of BERT$_{base}$ models. Lower is better.

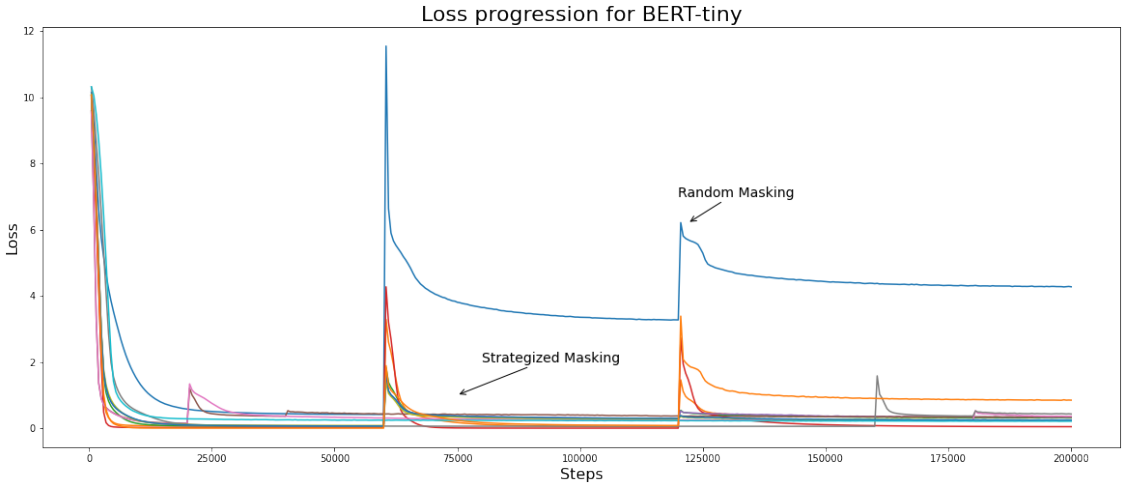


Figure 6.2: Loss during the pre-training of BERT$_{tiny}$ models. Lower is better.

6.2 Random Masking vs. Strategized Masking

Table 6.1 displays the results when applying four different masking strategies. The first was trained using the original Random Whole Word Masking, the second using the in this Thesis presented Strategized Masking. In addition, number three and four are two reductionist versions of Strategized Masking. One where only the lemmatization was used as a masking technique, and one where only the POS-based masking was used.

Masking strategy	**GLUE**	**CoLA**	**SST-2**	**MRPC**	**QQP**	**MNLI**	**MNLI-mm**	**QNLI**	**RTE**	**AX**
	Average	MCC	ACC	F1/ACC	F1/ACC	ACC	ACC	ACC	ACC	MCC
BERT-base										
Random Masking	48,6	0	82,3	79,1/68,9	~~0/82.4~~	62,5	63,5	60,3	51,9	21,9
Strategized Masking	51,5	11,2	82,2	77,9/68,1	56,4/80,8	66,3	36,5	60,2	51,8	**23,9**
Lemmatization only	48,2	11,6	~~49,9~~	78,4/68,4	57,6/81,5	~~31,2~~	67	62,6	52,3	0
POS-based Masking only	**57,1**	**20**	**87,5**	**80,5/69,9**	**60,8/84,8**	**72,7**	**71,6**	**67,1**	**53,5**	21,6
BERT-tiny										
Random Masking	**51,9**	8,7	82	~~79,9/66,5~~	54,0/**81,0**	60,9	**61,3**	58,7	50,3	12,6
Strategized Masking	50,9	0	81,6	~~79,9/66,5~~	**54,1**/80,3	59,4	60,5	59,1	51,7	12,1
Lemmatization only	51,8	**10,6**	81,1	~~79,9/66,5~~	53,4/80,8	60,4	59	58	51	12
POS-based Masking only	51,2	0	**82,4**	~~79,9/66,5~~	53,4/79,1	**63,2**	58,1	**61,4**	**52,2**	**20,2**

Table 6.1: Test set GLUE-results for different masking strategies. Higher is better applies to every task listed. Best results are in bold, striked out means the model failed to give meaningful predictions. Top half contains results for BERT_{base} and the bottom half contains results for BERT_{tiny}.

The overall results are very mixed. Both Random Masking and Strategized Masking are outperforming each other across different tasks. What is more interesting is that For BERT_{base}, the reduced version of Strategized Masking, POS-based masking, outperforms both Random Masking, the full Strategized Masking and the only lemmatization masking. In fact, it beats or matches some of the other best results across all experiments performed. Using only the lemmatization technique results in a significant loss in performance. Not only does it consistently score lower across most tasks, it fails entirely to train on the SST-2, MNLI and AX tasks.

For BERT_{tiny}, the best performance is shared between the POS-based masking and the Random Masking technique, although POS-based masking does have the majority of best peformances.

6.3 Effects of datasize

Table 6.2 shows the results when pre-training using different results from the corpus reduction.

Datasize	**GLUE** Average	**CoLA** MCC	**SST-2** ACC	**MRPC** F1/ACC	**QQP** F1/ACC	**MNLI** ACC	**MNLI-mm** ACC	**QNLI** ACC	**RTE** ACC	**AX** MCC
					BERT-base					
$\mathcal{C}_{100K}$	49,2	3,7	82,8	~~79,9/66,5~~	~~0/82.4~~	64,5	63,3	60,4	52,2	20,9
$\mathcal{C}_{1M}$	**54,6**	**14,1**	82,3	75,6/66,3	**60,5/84,2**	**69,1**	**68,2**	**65,2**	**52,8**	22,2
$\mathcal{C}_{10M}$	48,3	9,3	**83**	**77,9**/68,0	~~0/82.4~~	62,7	60,6	49,5	52,1	21,4
$\mathcal{C}_{100M}$	51,5	11,2	82,2	**77,9/68,1**	56,4/80,8	66,3	~~36,5~~	60,2	51,8	**23,9**
					BERT-tiny					
$\mathcal{C}_{100K}$	50,9	0	**82,2**	~~79,9/66,5~~	54,0/81,6	58,6	58,3	59,5	**51,9**	8,2
$\mathcal{C}_{1M}$	**51,7**	2,1	81,8	**80,0/66,7**	**56,3/83,3**	**60,9**	58,5	**63,4**	50,7	**12,8**
$\mathcal{C}_{10M}$	51,4	**7,5**	81,5	~~79,9/66,5~~	53,5/79,7	59,7	**60,8**	57,9	50,3	10,4
$\mathcal{C}_{100M}$	50,9	0	81,6	~~79,9/66,5~~	54,1/80,3	59,4	60,5	59,1	51,7	12,1

Table 6.2: Test set GLUE-results for models trained on different dataset versions from the corpus reduction. Higher is better applies to every task listed. Best results are in bold, striked out means the model failed to give meaningful predictions. Top half contains results for BERT$_{base}$ and the bottom half contains results for BERT$_{tiny}$.

For both BERT$_{base}$ and BERT$_{tiny}$ the models trained on the 1M-dataset perform the best on most tasks, sometimes by a considerable amount. For BERT$_{base}$, training a model with the 1M reduced corpus dataset amounts for a large lead in performance on MNLI and MNLI-mm.

6.4 Effects of tokenization strategies

Table 6.3 lists the results for model performance where every model is trained using a different tokenization techniques.

Tokenization strategy	**GLUE** Average	**CoLA** MCC	**SST-2** ACC	**MRPC** F1/ACC	**QQP** F1/ACC	**MNLI** ACC	**MNLI-mm** ACC	**QNLI** ACC	**RTE** ACC	**AX** MCC
					BERT-base					
Chunking	51,5	11,2	82,2	77,9/68,1	56,4/80,8	66,3	~~36,5~~	60,2	51,8	**23,9**
Truncation	**57,1**	**17,4**	**85,7**	**79,6/69,7**	**61,8/85,9**	**72,7**	**72,9**	**69,4**	**54,7**	21,4
					BERT-tiny					
Chunking	50,9	0	81,6	~~79,9/66,5~~	**54,1/80,3**	59,4	60,5	**59,1**	51,7	12,1
Truncation	**51**	0	**81,7**	~~79,9/66,5~~	53,7/80,3	**61**	**60,8**	**59,1**	**52,1**	**17,8**

Table 6.3: Results for training models with different tokenization techniques. Higher is better applies to every task listed. Best results are in bold, striked out means the model failed to give meaningful predictions. Top half contains results for BERT$_{base}$ and the bottom half contains results for BERT$_{tiny}$.

For BERT$_{base}$, sequences which were truncated obtain some of the best performances across the table, with chunking being nowhere near that performance. For BERT$_{tiny}$ the performance of truncation will either match or beat the improve of chunking.

6.5 Effects of split-training

Table 6.4 contains the results for training with a different distribution of steps for split-training. For BERT$_{base}$, the steps total to 100k. For BERT$_{tiny}$, the steps total to 200k. This means that for "No split-training" the total number of steps was used in 1 go for each model respectively.

Steps distribution	**GLUE**	**CoLA**	**SST-2**	**MRPC**	**QQP**	**MNLI**	**MNLI-mm**	**QNLI**	**RTE**	**AX**
	Average	MCC	ACC	F1/ACC	F1/ACC	ACC	ACC	ACC	ACC	MCC
BERT-base										
30K/30k/40k	51,5	11,2	82,2	77,9/68,1	56,4/**80,8**	66,3	36,5	60,2	51,8	**23,9**
10k/10k/80k	52,7	10,8	83,2	77,0/67,2	54,8/79,4	62,2	64,4	61,1	51,9	19,9
10k/80k/10k	52,3	7,2	82,5	~~79,9/66,5~~	**56,7/80,8**	62,4	60,6	60,7	52,2	20,6
80k/10k/10k	52,9	8,5	83,3	~~79,9/66,5~~	55,5/80,3	64,7	64,9	61,7	51,7	20,1
No split-training	**53**	**17,4**	**86,8**	**80,1/70,1**	~~0/82.4~~	**73,6**	**72,8**	**65,8**	**52,3**	23,4
BERT-tiny										
60k/60k/80k	**50,9**	0	81,6	~~79,9/66,5~~	54,1/80,3	59,4	60,5	59,1	51,7	12,1
20k/20k/160k	50,8	0	**83,1**	~~79,9/66.5~~	**54,2**/80.0	60,1	58,3	**59,3**	50,7	13,5
20k/160k/20k	50,1	0	82	**77,3/68,0**	53,7/**80,4**	57	55,6	57,6	50,3	6,8
160k/20k/20k	50,7	0	81,1	~~79,9/66,5~~	53,6/79,5	59,8	59,1	58,7	52	11,4
No split-training	**50,9**	0	82	~~79,9/66,5~~	53,5/79,7	**60,3**	**60,6**	58,3	**52,1**	**14,2**

Table 6.4: Results for training models with a different amount of steps during different sequence lengths. Higher is better applies to every task listed. Best results are in bold, striked out means the model failed to give meaningful predictions. Top half contains results for BERT_{base} and the bottom half contains results for BERT_{tiny}.

For both BERT_{base} and BERT_{tiny}, the distribution in steps seems to make no large difference in performance. What is apparent is that performing no split-training leads to the highest performances when using BERT_{base}. For BERT_{tiny} the overall GLUE-score is the same for the default steps distribution and no split-training. There are also tasks where split-training does obtain a higher score but running no split-training still has the majority of best performances.

For BERT_{base} the models using 10k/10k/80k and 80k/10k/10k fail to obtain a best performance on any task. In addition, all BERT_{tiny} models fail to fine-tune on the CoLA task.

6.6 Loss benchmark

Table 6.5 and Table 6.6 display the performance of BERT_{base} and BERT_{tiny} respectively on the loss benchmark. Results are ordered by type of experiment but grouped for overall convenience.

For BERT_{base}, the first noticeable thing is that due to the batch size and limited number of steps, some models have not seen a full epoch on split 3 of the training data. This is not the case for BERT_{tiny} which, due to its smaller model size, allowed for a larger batch size.

The BERT_{base} models also show that for the Random Masking benchmark, all models trained on Strategized Masking match or beat the loss of the model trained specifically through Random Masking. The only exception being the Lemmatization only masker which also showed poor performance on the GLUE benchmark. For the Strategized Masking benchmark, the loss is equal or better than the model trained through Random Masking. This is especially interesting because most of the models trained through Strategized Masking have not seen a full epoch of their original training data.

For BERT_{tiny}, the performance is on the Random Masking benchmark is very equal across all models. One exception here being the model trained on $\mathcal{C}_{1M}$ which shows a slightly higher loss. The best loss here is almost of negligible distance to other models. The performance on Strategized Masking shows the exact converse, $\mathcal{C}_{1M}$ obtains a significant improvement over other models.

Experiment type	Experiment	RandomMasking	StrategizedMasking	Epochs
Masking	Random Masking	2.71e+06	3.57e+07	4.31
	Strategized Masking	2.75e+06	3.34e+07	0.34
	Lemmatization only	4.04e+06	8.69e+06	10.78
	POS-based Masking only	**1.93e+06**	2.01e+06	0.94
Datasize	$\mathcal{C}_{100K}$	2.78e+06	3.35e+07	96.39
	$\mathcal{C}_{1M}$	2.77e+06	1.8e+06	9.19
	$\mathcal{C}_{10M}$	2.79e+06	3.35e+07	0.92
	$\mathcal{C}_{100M}$	2.75e+06	3.34e+07	0.34
Tokenization	Chunking	2.75e+06	3.34e+07	0.34
	Truncation	1.98e+06	**8.15e+05**	1.56
Steps distribution	30k/30k/40k	2.75e+06	3.34e+07	0.34
	10k/10k/80k	2.75e+06	3.34e+07	0.69
	10k/80k/10k	2.75e+06	3.38e+07	0.09
	80k/10k/10k	2.77e+06	3.35e+07	0.09
	No split-training	1.99e+06	8.38e+05	2.09

Table 6.5: Results for BERT$_{base}$ models. Lower is better. The best result is now selected from the entire table and is given in bold.

Experiment type	Experiment	RandomMasking	StrategizedMasking	Epochs
Masking	Random Masking	**3.04e+06**	3.83e+07	34.48
	Strategized Masking	3.06e+06	3.82e+07	2.76
	Lemmatization only	3.05e+06	3.81e+07	34.48
	POS-based Masking only	3.05e+06	3.8e+07	3.01
Datasize	$\mathcal{C}_{100K}$	3.05e+06	3.8e+07	769.23
	$\mathcal{C}_{1M}$	3.24e+06	**2.29e+06**	73.53
	$\mathcal{C}_{10M}$	3.06e+06	3.81e+07	7.39
	$\mathcal{C}_{100M}$	3.06e+06	3.82e+07	2.76
Tokenization	Chunking	3.06e+06	3.82e+07	2.76
	Truncation	3.05e+06	3.8e+07	4.98
Steps distribution	30k/30k/40k	3.06e+06	3.82e+07	2.76
	10k/10k/80k	3.06e+06	3.81e+07	5.52
	10k/80k/10k	3.05e+06	3.81e+07	0.69
	80k/10k/10k	3.05e+06	3.81e+07	0.69
	No split-training	3.06e+06	3.81e+07	8.34

Table 6.6: Results for BERT$_{tiny}$ models. Lower is better. The best result is now selected from the entire table and is given in bold.

6.7 Discussion

6.7.1 Reflection on results

With these results in mind, it is time to look at the implication of the results in accordance to the 4 concepts introduced in this Thesis.

First, the obtained results it seems that Strategized Masking is an improvement compared to Random Masking. Earlier it was hypothesized that this would force models to learn more linguistic structure compared to guessing randomly masked tokens. Surprisingly, the lemmatization component in Strategized Masking seemed to significantly hurt performance. Running the lemmatization only component obtained one of the worst scores across the board. This is likely caused by the lack of variation in masks it has to guess. Lemmatization only affects basic nouns or adjectives, which may make up too little of a language component for the model to learn well from. The failure to learn from lemmatization may also stem from how vocabularies are constructed. Literature provides us with many different forms of vocabulary building. Three mainstream approaches for the English language are: Byte-Pair Encoding [26], WordPiece [24] and Unigram [12]. The practical interpretation is that words are often added on frequency or some statistical dependency. This form of vocabulary construction may be a problem, because lemmatization in the English language is often done by stripping words of their end characters. The word "tigers" is a plural, and its lemma is "tiger". The problem of learning from this lemmatization occurs when both words are in the vocabulary, instead of being tokenized as "tiger" + "##s". The same problem may apply to verbs, where the word "walking" is stemmed to "walk". One would prefer it if these words were tokenized as "walk"+"##ing". This means that the poor performance of the lemmatization only component may be caused by the lack of grammatical structure of the vocabularies used.

Second, the results show that there is a desire for models to be trained on non-redundant datasets. The models trained on $\mathcal{C}_{1M}$ and $\mathcal{C}_{10M}$ significantly outperform the other datasets. It is important to remember that $\mathcal{C}_{1M}$ and $\mathcal{C}_{10M}$ only contained 87,2% and 99,2% of the unique tokens in $\mathcal{C}_{100M}$ respectively. This while it took a factor 28 and 2,8 less data to do so. This means that there is some evidence that pre-training would benefit from data which is less redundant without losing variety. There is a bit of pushback here, as recent research finds that some key NLU of the SuperGLUE benchmark are not learned [39] without using a few billion words of pre-training data. Their findings are not mutually exclusive with the experiments done in this project. A possible explanation could be that the 30B datasets introduce some new linguistic variety that the model benefits from. It would then be the challenge to maintain this variety under a corpus reduction.

Third, the results for tokenization strategies are very clear regarding resulting performance. The introduced method of chunking significantly hurts performance across almost all tasks and for both model sizes. It may be that the spaCy's Sentence Segmentizer introduces certain biases which matter during pre-training. Looking back at Figure 4.2, it may be that chunk 2 and 3 are missing important contextual information from prior chunks.

Fourth, was split-training valuable? For BERT_{base} the scores are very similar across different step distributions. Nonetheless, running no split-training whatsoever significantly outperformed other distributions. For BERT_{tiny} the results are more spread around the different step distributions. This hints towards a strong influence of model size on this performance. Regardless, in terms of performance it does not seem valuable to introduce split-training as a concept. Split-training was introduced in chapter 4 to replace the gluing together of sequences which may end up being too long and needing to be truncated. However, chunking also significantly hurt performance compared to truncation. Therefore there seems to be no value in warm-starting the pre-training of models using shorter and simpler language using split-training.

Last, there is the performance on the loss benchmark. BERT_{base} shows the most interesting

results. All models except the Random Masking one are pre-trained using Strategized Masking. All models either beat or closely match the performance of Random Masking on both benchmarks. This means that Strategized Masking makes a model more competent at guessing masks, both random and strategized masked. It also means the converse is not true, Random Masking does not make the model competent at being able to guess strategized masks.

6.7.2 Computational resources

One the challenges imposed upon this Thesis was to limit the amount of resources to be used. This meant using smaller models, a smaller dataset and lower number of steps than commonly used in state-of-the-art producing research. This also lead to computational efficiency being one of the core aspects of this project. However, this also meant that it was not possible to process and pre-train with the full unreduced corpus, which would have been a nice baseline.

In the literature for BERT, ELECTRA or StructBERT, using 1M training steps seems the norm rather than the exception. This project had to do with less than a tenth of that. Their hardware availability also meant being able to use far bigger batch sizes: BERT used a batch size of 256, 8 times bigger than what was used in this Thesis. Had it been decided to run the pre-training at the same level would have required ∼2 weeks of 24/7 pre-training for every model. The pre-training of a single model would not have been a problem when looking at the overall duration of this Thesis. However, it would have made a lot of the comparison experiments conducted impossible due to the usage limits of Google Colab.

As expected, this sometimes reflects in the results, e.g. BERT_{tiny} models are consistently unable to perform anything on CoLA or MRPC. It seems that BERT_{tiny} has an inherent learning limit when learning from scratch because of its size, given how equal the loss benchmark results are and how it consistently failed to fine-tune on CoLA. This can be largely explained that BERT_{tiny} was designed to be more efficient with minimal loss of performance through knowledge distillation [11] rather than the intention to be pre-trained from scratch.

However, more models (including BERT) suffer from failing to fine-tune on downstream tasks [5] [18]. In fact, it is not hard to find entries on the GLUE benchmark with the same scores as presented here, without any mention that these scores are due single class predictions on an imbalanced test set. The original paper of GLUE [35] mentions that this is considered the baseline but it seems counter-intuitive to have it represented by a majority classifier.

Regardless, this failure to fine-tune can be largely explained by 3 factors: (1) the data to fine-tune on is for some tasks very limited and linguistically complex (e.g. WNLI only has 635 sequences for fine-tuning and has a biased training and validation set which creates a mismatch towards the test set); (2) hyperparameter search is computationally expensive, meaning it can be hard to find a setting that works; (3) the pre-training was too short which did not allow for a good ingraining of the training data into the model parameters. With these factors in mind, is it important to note that the best task performance across all models is comparable to that of low/mid-level performing models on the GLUE-benchmark.

The pre-trained models were able to perform quite well given the restrictions that were put upon them, specifically the combination of model size and limited number of training steps. However, it is unfortunate that it was not possible to pre-process full Project Gutenberg corpus, leaving out a comparison between a full unreduced corpus and a reduced one. This does not deteriorate the current comparison, given that $\mathcal{C}_{1M}$ still consistently outperforms $\mathcal{C}_{100M}$ while using a factor 28 less tokens.

Chapter 7

Future work & Conclusion

7.1 Future work

Based on the succesful work covered in this Thesis, several entry points for future research will be provided.

The first is an extension to Strategized Masking. The basics of the concept were to introduce a better way of having the model perform the Cloze task as mentioned in chapter 2. But there are other extensions possible which tie into the improvements made by StructBERT [36]. Along with masking, StructBERT shuffles around words in a tri-gram. Like Random Masking, this is essentially reordering words into a random order. A more directed way would be to shuffle around specific Parts-Of-Speech pairs into a different order. For example, the sequence "The blue elephant" contains an adjective-verb pair: "blue-elephant". This can be swapped around to "The elephant blue". Like POS-based masking, this POS-based shuffling could augment the base technique. More likely, there are many other techniques possible that can enhance the masking procedure to have the model guess specific linguistic structures.

The second is that of Corpus Reduction. This work shows that there is indeed value in filtering linguistic variety from a very large corpus. The method presented in this Thesis can be one of the stepping stones towards a standardized pre-training corpus, perhaps even multiple with different sizes. It would allow for research which is more reproducible, but also be able to contain much more diverse language and topics. The advance made by most models is due to other factors rather than an increase in data [39]. For comparison, despite their discovery being 5 years apart both word embeddings and BERT used billions of words to pre-train on, but it is much more likely that the sharp increase in performance between them came from changes in architecture, transfer-learning and different pre-training approaches.

The third would be a reproduction of this work but in combination with ELECTRA and a larger computational budget. The choice for ELECTRA here is because ELECTRA provided improvements to the overall pre-training process. For comparison, the model "StructBERT + CLEVER" is currently the second highest model which obtains the majority of its performance due to the focus on fine-tuning with the parameter description reading: "fine-tuning StructBERT on MNLI before training on MRPC, RTE and STS-B". Based on the work in this Thesis, it is apparent that there are still opportunities to improve the pre-training process and that fine-tuning optimizations should come afterwards.

The fourth and final one is a more open and undirected entry point, which is the formation of a vocabulary which includes grammatical structure. This was briefly touched upon section 6.7. The lack of this grammatical structure in vocabularies can be seen as an artifact of their mainstream construction methods. The advantage are of using these existing methods are very clear, it allowed

for the intelligent formation of a large scale vocabulary without requiring the need for extensive supervision. Supervision would be very expensive, and even more problematic for multilingual models where the vocabulary covers cross-language tokens. Instead, the construction methods rely on word frequency or another form of statistical dependencies. However, this leads to words losing their intrinstic relationship that is present through grammar. The words "walk" and "walking" are highly related but the way that vocabulary is practically implemented does not allow to account for that fact. POS-based masking significantly improved performance due to its nature of targeted learning. Lemmatization thrives on this same narrative, but the current implementation leads to a significant loss in performance due to the inability to use its share of targeted learning. Overall, this shows that there is a serious desire for a method of vocabulary construction which integrates grammatical structure.

7.2 Conclusion

This Thesis set out to answer 3 research questions: (1) What are methods that allow for NLP-models to be pre-trained in a targeted manner? (2) How can a large corpus which spans multiple human lifetimes be distilled into a smaller one without losing linguistic diversity? (3) What are strategies that can minimize the number of lost tokens when dealing with sequences that exceed the maximum length of the model input?

Chapter 2 introduced Strategized Masking, which used a Part-Of-Speech tagger and Lemmatizer for a new masking strategy. The loss-progression during pre-training and subsequent performance on the GLUE-benchmark shows that POS-based masking augments the pre-training process beyond that off the standard Random Masking which is so prevalent in mainstream literature. From the loss-benchmark, it was shown to be a more successful way of learning targeted linguistic features than Random Masking.

Chapter 3 introduced a PTAS for corpus reduction. Through the GLUE-benchmark, it was shown that there is indeed value in using less redundant data during pre-training.

Chapter 4 introduced two concepts: chunking and split-training. Chunking had the specific goal to preserve tokens which would be lost on the tokenization of too long sequences. This turned out to significantly hurt performance, most likely from the missing context. Split-training was introduced as an alternative to gluing sequences together which could once again end up being too long, while also increase the training efficiency from short sequences. This also turned out to be significantly hurting performance, this time with the possible reason being left open.

Several entries for future work are presented and it is hoped that these methods will augment existing pre-training procedures.

Bibliography

[1] Sriram Balasubramanian, Naman Jain, Gaurav Jindal, Abhijeet Awasthi, and Sunita Sarawagi. What's in a name? are BERT named entity representations just as good for any other name? In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 205–214, Online, July 2020. Association for Computational Linguistics. 1

[2] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. 1

[3] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2020. 3, 26, 27, 28

[4] George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–288, 1957. 14

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. 1, 2, 3, 10, 11, 12, 25, 26, 35

[6] Stephen Fagan and Ramazan Gençay. An introduction to textual econometrics. *Handbook of empirical economics and finance*, pages 133–154, 12 2010. 2

[7] Yoav Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015. 1, 5

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. 4

[9] Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018. 6

[10] Jennifer Hu, Jon Gauthier, Peng Qian, Ethan Wilcox, and Roger Levy. A systematic assessment of syntactic generalization in neural language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1725–1744, Online, July 2020. Association for Computational Linguistics. 3

[11] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling BERT for natural language understanding. *CoRR*, abs/1909.10351, 2019. 35

[12] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. *CoRR*, abs/1804.10959, 2018. 34

[13] Y. Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019. 1, 3, 11, 12, 25, 26

[14] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *CoRR*, abs/1708.00107, 2017. 5

[15] Vincent Micheli, Martin d'Hoffschmidt, and François Fleuret. On the importance of pre-training data volume for compact language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7853–7858, Online, November 2020. Association for Computational Linguistics. 3

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. 5

[17] Ines Montani, Matthew Honnibal, Matthew Honnibal, Sofie Van Landeghem, Adriane Boyd, Henning Peters, Maxim Samsonov, Jim Geovedi, Jim Regan, György Orosz, Paul O'Leary McCann, Duygu Altinok, Søren Lind Kristiansen, Roman, Leander Fiedler, Grégory Howard, Wannaphong Phatthiyaphaibun, Yohei Tamura, Explosion Bot, Sam Bozek, murat, Mark Amery, Björn Böing, Pradeep Kumar Tippa, Leif Uwe Vogelsang, Ramanan Balakrishnan, Vadim Mazaev, GregDubbin, jeannefukumaru, and Walter Henry. *explosion/spaCy: v3.0.5: Bug fix for thinc requirement*, March 2021. 8

[18] Timothy Niven and Hung-Yu Kao. Probing neural network comprehension of natural language arguments. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4658–4664, Florence, Italy, July 2019. Association for Computational Linguistics. 1, 35

[19] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010. 5

[20] Steven Piantadosi. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21, 03 2014. 11

[21] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*, 2019. 3, 11, 18

[22] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. 1

[23] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A Primer in BERTology: What We Know About How BERT Works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 01 2021. 1

[24] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152, 2012. 34

[25] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *CoRR*, abs/1907.10597, 2019. 1

[26] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909, 2015. 34

[27] Koustuv Sinha, Robin Jia, Dieuwke Hupkes, J. Pineau, Adina Williams, and Douwe Kiela. Masked language modeling and the distributional hypothesis: Order word matters pre-training for little. *ArXiv*, abs/2104.06644, 2021. 1, 2, 8, 11

[28] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. *CoRR*, abs/1906.02243, 2019. 1

[29] Wilson L. Taylor. "cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, 30(4):415–433, 1953. 6

[30] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962, 2019. 4

[31] Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. *ArXiv*, abs/1908.08962, 2019. 25

[32] Marten van Schijndel, Aaron Mueller, and Tal Linzen. Quantity doesn't buy quality syntax with neural language models. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5831–5837, Hong Kong, China, November 2019. Association for Computational Linguistics. 3

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. 6, 17

[34] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. *arXiv preprint 1905.00537*, 2019. 1

[35] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR. 1, 26, 35

[36] Wei Wang, Bin Bi, Ming Yan, Chen Wu, Jiangnan Xia, Zuyi Bao, Liwei Peng, and Luo Si. Structbert: Incorporating language structures into pre-training for deep language understanding. In *International Conference on Learning Representations*, 2020. 1, 3, 10, 36

[37] Alex Warstadt, Yian Zhang, Xiaocheng Li, Haokun Liu, and Samuel R. Bowman. Learning which features matter: RoBERTa acquires a preference for linguistic generalizations (eventually). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 217–235, Online, November 2020. Association for Computational Linguistics. 3

[38] Yonghui Wu, M. Schuster, Z. Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, M. Krikun, Yuan Cao, Qin Gao, Klaus Macherey, J. Klingner, Apurva Shah, M. Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Y. Kato, Taku Kudo, H. Kazawa, K. Stevens, George Kurian, Nishant Patil, W. Wang, C. Young, Jason R. Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, G. Corrado, Macduff Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144, 2016. 6, 12

[39] Yian Zhang, Alex Warstadt, Haau-Sing Li, and Samuel R. Bowman. When do you need billions of words of pretraining data? *CoRR*, abs/2011.04946, 2020. 3, 34, 36

[40] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *arXiv preprint arXiv:1506.06724*, 2015. 11