https://lib.uliege.be https://matheo.uliege.be

Master's Thesis : NetBERT: A Pre-trained Language Representation Model for Computer Networking

Auteur : Louis, Antoine
Promoteur(s) : Louppe, Gilles
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en science des données, à finalité spécialisée
Année académique : 2019-2020
URI/URL : http://hdl.handle.net/2268.2/9060

Avertissement à l'attention des usagers :

Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.

Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.

University of Liège - Faculty of Applied Sciences
In collaboration with Cisco Systems

NetBERT: A Pre-trained Language Representation Model for Computer Networking

Advisor:
Prof. G. Louppe

Jury:
Prof. G. Louppe
Prof. P. Geurts
Prof. L. Mathy
H. De Pra

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Science in Data Science & Engineering

by Antoine Louis

Academic year 2019-2020

Abstract

Obtaining accurate information about products in a fast and efficient way is becoming increasingly important at Cisco as the related documentation rapidly grows. Thanks to recent progress in natural language processing (NLP), extracting valuable information from *general domain* documents has gained in popularity, and deep learning has boosted the development of effective text mining systems. However, directly applying the advancements in NLP to *domain-specific* documentation might yield unsatisfactory results due to a word distribution shift from general domain language to domain-specific language. Hence, this thesis aims to determine if a large language model pre-trained on domain-specific (*computer networking*) text corpora improves performance over the same model pre-trained exclusively on general domain text, when evaluated on in-domain text mining tasks.

To this end, we introduce NetBERT (Bidirectional Encoder Representations from Transformers for Computer Networking), a domain-specific language representation model based on BERT (Devlin et al., 2018) and pre-trained on large-scale computer networking corpora. Through several extrinsic and intrinsic evaluations, we compare the performance of our novel model against the general-domain BERT. We demonstrate clear improvements over BERT on the following two representative text mining tasks: networking *text classification* (0.9% F1 improvement) and networking *information retrieval* (12.3% improvement on a custom retrieval score). Additional experiments on *word similarity* and *word analogy* tend to show that NetBERT capture more meaningful semantic properties and relations between networking concepts than BERT does. We conclude that pre-training BERT on *computer networking* corpora helps it understand more accurately domain-related text.

Acknowledgements

First and foremost, I would like to express my special thanks of gratitude to my advisor Gilles Louppe, who has always been present at the slightest of my concerns, and who has ensured a more than regular follow-up of my work, giving me constructive feedback and valuable advice at every stage of my thesis.

Next, I would like to thank more than generously Hugues De Pra, who gave me the golden opportunity to do this wonderful internship at Cisco, and made me feel like an integral part of his team from day one, by including me in meetings and events, introducing me to a very large number of employees, and keeping in regular and interested contact with me.

Thank you to Dmitry Goloubew, who gave me the chance to work on a subject that interests me immensely (state-of-the-art NLP models), putting at my disposal an impressive computing power, and allowing me a very pleasant freedom in my research while giving me his expert opinion on some interesting directions to explore.

Thank you to Antoine Wehenkel and Tom Crasset for useful feedback on the manuscript.

Finally, thank you to my family, friends and loved ones for always helping me find a balance between work and life.

Contents

Abstract **i**

Acknowledgements **ii**

1 Introduction **1**
1.1 The Cisco Search Problem . . . 1
1.2 Text Mining . . . 2
1.3 Research Questions . . . 3
1.4 Thesis Outline . . . 3

2 Related Work **5**
2.1 A Brief History of Natural Language Processing . . . 5
2.1.1 NLP before the Deep Learning Era . . . 5
2.1.2 NLP in the Deep Learning Era . . . 7
2.2 Word Embedding . . . 9
2.2.1 Count-based Models . . . 10
2.2.2 Prediction-based Models . . . 11
2.2.3 Deep Contextual Models . . . 14
2.3 Domain-specific BERT-based Models . . . 15

3 BERT **16**
3.1 Self-attention . . . 16
3.1.1 Multi-head Attention . . . 19
3.2 Model . . . 20
3.2.1 Architecture . . . 20
3.2.2 Input Representations . . . 21
3.2.3 Parameters . . . 23
3.3 Pre-training Procedure . . . 24
3.3.1 Masked Language Modeling . . . 24
3.3.2 Next Sentence Prediction . . . 26
3.4 Downstream Tasks . . . 26
3.4.1 Fine-tuning Approach . . . 26
3.4.2 Feature-based Approach . . . 27

4 NetBERT **29**
4.1 Model Choice . . . 29
4.1.1 The Google Search Update . . . 29
4.1.2 BERT as Knowledge Base . . . 30
4.1.3 Model Size . . . 30
4.2 Pre-training Data . . . 32
4.2.1 Data Collection . . . 32
4.2.2 Data Preparation . . . 32

4.2.3 Processed Dataset . . . 33
4.3 Pre-training . . . 34
4.3.1 Strategies . . . 34
4.3.2 Setup . . . 36
4.3.3 Results . . . 36

5 Experiments **39**
5.1 Text Classification . . . 39
5.1.1 Dataset . . . 40
5.1.2 Fine-tuning . . . 41
5.1.3 Results . . . 42
5.1.4 Further Analysis . . . 43
5.1.5 Conclusions . . . 49
5.2 Information Retrieval . . . 50
5.2.1 Dataset . . . 50
5.2.2 Search Score . . . 52
5.2.3 Results . . . 53
5.2.4 Further Analysis . . . 55
5.2.5 Conclusions . . . 57
5.3 Word Similarity . . . 57
5.3.1 Task Description . . . 57
5.3.2 Evaluations . . . 58
5.3.3 Conclusions . . . 60
5.4 Word Analogy . . . 60
5.4.1 Task Description . . . 60
5.4.2 Evaluations . . . 61
5.4.3 Conclusions . . . 63

6 NetBERT Search Engine **64**
6.1 Implementation . . . 64
6.1.1 Index Creation . . . 64
6.1.2 Real-time Search . . . 65
6.2 Applications . . . 66
6.2.1 Cisco Corpus . . . 66
6.2.2 RFC Corpus . . . 66
6.3 Limitations and Possible Improvements . . . 67
6.3.1 Sentence Representation . . . 67
6.3.2 Speed and Memory . . . 68

7 Conclusions **69**
7.1 Future Directions . . . 70

Appendices **71**

A Additional Details about BERT **72**
A.1 Model Parameters . . . 72
A.2 Model Architecture . . . 74
A.3 BERT in Google Search . . . 75

B Additional Resources about the Experiments **76**
B.1 Text Classification . . . 76
B.1.1 Classification Performance Metrics . . . 76
B.1.2 Optimal Hyperparameters Search . . . 78

C About Cisco Systems **80**

Bibliography **80**

Chapter 1

Introduction

Language played an enormous role in the development of the human race. It is co-operation through interaction and communication that made Homo Sapiens the first, fully dominant species in Earth's history. Over time, we progressively realized that social intelligence equaled power. The reason we have power over tigers is not because we have bigger muscles or sharper teeth, it is because we are able to communicate in detail a well thought-out tactic to defeat it, using complex human language. According to the two-time winner of the Pulitzer Prize Edward O. Wilson (2014), it is our increasing ability to communicate, recognise, evaluate, co-operate and compete that rapidly took us to the top of the food chain.

Today, the digital era allows us to share an impressive amount of knowledge expressed in human language through books, articles, blog posts, podcasts and videos, available online anywhere in the world to anyone with an internet connection. Because most of this knowledge is stored and accessed via computer systems, we have seen a growing interest in the development of methods allowing humans to communicate, or at least be understood by machines.

In this thesis, we primarily focus on such methods, also referred as *natural language processing* (NLP) techniques. Specifically, we develop a model which is able to process domain-specific language, by learning in a self-supervised way from a huge in-domain text corpus. The goal of this thesis is then to study if that model leads to a better understanding of the domain-specific language, compared to a model that learned from general-domain language. More information about this research question is presented in Section 1.3. But first, Section 1.1 introduces one of the challenges encountered at Cisco, and which mainly motivates this work. Then, Section 1.2 describes the current state-of-the-art approach for text mining. Finally, Section 1.4 outlines the structure of this thesis.

1.1 The Cisco Search Problem

Cisco is the worldwide leader in IT, networking, and cybersecurity solutions. They help companies of all sizes transform how people connect, communicate, and collaborate. To this end, Cisco offers a portfolio of more than 650 products in a variety of areas: networking, wireless and mobility, security, analytics, video, internet of things (IoT), and many others.[1] As shown in Figure 1.1, 75% of the company's turnover comes from the sales of these products (i.e., 39 billion dollars in 2019) the remaining 25% coming from their services such as technical support, digital training, and business optimization.

Cisco products are sold by Systems Engineers, whose job is to understand the need of a customer, find the appropriate product, and sell that solution to the client. Their work requires an extensive knowledge of the products that are being sold, as well as the underlying technologies. Frequently, Systems Engineers have to face technical engineers who don't hesitate to ask metic-

[1] https://www.cisco.com/c/en/us/products/a-to-z-series-index.html

ulous questions in order to judge whether the proposed product is suitable for their problem or not.

Figure 1.1: Financial highlights for Cisco fiscal year 2019.[2]

Like every human being, Systems Engineers are not infallible databases. Memory is sometimes lacking. They may not be aware of the exact functioning of a product or the latest changes after a software update. They could face questions to which the answers are not obvious, and which require a deep dive into the documentation. That is why a large part of their job actually consists in looking for the right information in the thousands and thousands of documents describing products, technologies and standards. For now, most employees perform their search using Cisco's internal search engine, which is based on a simple instance of the well-known *Elasticsearch*. According to the public notice, the latter is not satisfactory at all. The ranking of the results is often very bad and these results are mostly entire documents that can sometimes be several thousand pages long. Unsurprisingly, the search process is tiresome and frustrating. Consequently, there is a strong desire among the employees for accurate text mining tools in order to extract specific information from the documentation.

To give an order of magnitude, the Sales and Marketing Department represents about 34% of employees working at Cisco in 2019 (i.e., 26,200 people).[3] By improving the way they retrieve information, we improve the productivity of thousands of people, who then spend less time searching for answers and more time helping customers.

1.2 Text Mining

By definition, text mining refers to the process of extracting interesting information and knowledge from unstructured text (Hotho et al., 2005). In the past years, text mining models have been greatly improved by the advancements of deep learning techniques used in NLP. Recently, major progress has been driven by the adoption of pre-trained language models such as ELMO (Peters et al., 2018), GPT (Radford et al., 2018) or BERT (Devlin et al., 2018). These models all rely on a two-step learning approach. First, they learn contextual word representations from a large amount of text in a self-supervised way. This stage is commonly referred as the *pre-training*. Then, these pre-trained language representations can be applied to downstream NLP tasks by choosing between two supervised learning strategies: *feature-based* and *fine-tuning*. The

[2] https://www.cisco.com/c/dam/en_us/about/annual-report/cisco-annual-report-2019.pdf
[3] https://www.statista.com/statistics/350534/cisco-employees-by-line-item/

feature-based approach uses another task-specific architecture that include the pre-trained representations as additional features for learning the given task. In contrast, the fine-tuning approach introduces minimal task-specific parameters and is trained on the downstream tasks by simply fine-tuning all pre-trained parameters. Arguably, the main advantage of such models comes from their large-scale self-supervised pre-training, i.e., their ability to learn word representations from large unannotated text corpora. This is especially beneficial when learning a domain-specific language, where annotated data is difficult and expensive to collect due to the expertise required for quality annotations. With pre-trained language models, labeled data is only needed in small quantity for the supervised learning step on downstream tasks. Recently, these models have shown to be effective for improving many sentence-level tasks, including natural language inference (NLI) and semantic textual similarity (STS).

Despite these impressive advancements, directly applying state-of-the-art NLP methodologies to domain-specific text mining might have some limitations. Indeed, recent word representation models are pre-trained and tested mainly on general-domain datasets such as Wikipedia, which might badly reflect the targeted word distribution and thus not be as performing when applied to domain-specific text. This work hypothesizes that current state-of-the-art word representation models need to be pre-trained on domain-specific corpora to be effective in related mining tasks.

1.3 Research Questions

The main research question this thesis aims to answer is the following:

> Does a large language representation model pre-trained on domain-specific text improve performance over the same model pre-trained on general-domain corpora, when evaluated on domain-specific text mining tasks?

In order to find an answer to this research question, multiple minor equally interesting research questions have to be considered. They can be summarized as follows:

- Which language representation model should be considered for the purpose of domain-specific text mining given limited computational resources?
- What are some good pre-training strategies to consider in order to achieve the best possible model with one unique run, given the huge computational time related to the pre-training of current state-of-the-art language representation models?
- How can language representation models be efficiently evaluated on domain-specific tasks, given little or no labelled data?

1.4 Thesis Outline

The rest of the thesis is structured as follows:

- **Chapter 2, Related Work.** This chapter briefly introduces the big steps in NLP research, from the creation of the field in the 1940s until the latest state-of-the-art models nowadays. Then, it dives into the concept of *word embeddings* and some common methods used to create them. Finally, it briefly reviews existing approaches that use BERT for domain-specific problems.
- **Chapter 3, BERT.** This chapter explains BERT (Devlin et al., 2018), the language representation model which is at the core of this thesis. It describes in detail the self-attention mechanism, the model architecture, its pre-training procedure and application to downstream tasks.

- **Chapter 4, NetBERT.** This chapter covers the pre-training of NetBERT, our novel language representation model pre-trained on large-scale computer networking corpora. It starts by motivating the choice of using BERT for this work. Then, it describes the collection and processing of the large text corpus used for pre-training the novel model. Finally, it summarizes some robust pre-training strategies gathered from the literature, as well as the pre-training setup and results.

- **Chapter 5, Experiments.** This chapter introduces several experiments on which NetBERT is extrinsically and intrinsically evaluated and compared to BERT. These include computer networking *text classification*, *information retrieval*, *word similarity* and *word analogy*.

- **Chapter 6, NetBERT Search Engine.** This chapter explains the implementation of a similarity-based search engine that uses NetBERT embeddings for information retrieval. It then describes its application to two different networking corpora. Finally, it discusses its limitations and some possible improvements to the current system.

- **Chapter 7, Conclusions.** This chapter summarizes the main contributions of this work. It then concludes the thesis with a brief discussion about the potential future directions and applications for NetBERT.

Chapter 2

Related Work

This chapter provides a detailed introduction to previous related work. First, Section 2.1 introduces the field of *natural language processing*, and describes the big innovations from the creation of the domain to the current state-of-the-art models. Then, Section 2.2 explains the concept of *word embedding* and discusses some popular word embedding models. Finally, Section 2.3 briefly reviews BERT-based models that have been pre-trained on domain-specific corpora, similarly to our work.

2.1 A Brief History of Natural Language Processing

Natural language processing (NLP) is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis (Liddy, 2001). The purpose of these techniques is to achieve human-like language processing for a range of tasks or applications. Although it has gained enormous interest in recent years, research in NLP has been going on for several decades dating back to the late 1940s. This review divides its history into two main periods: NLP before and during the deep learning era.

2.1.1 NLP before the Deep Learning Era

Figure 2.1: The big stages of NLP before the deep learning era.

It is generally agreed that Weaver's memorandum (Shannon and Weaver, 1949) brought the idea of the first computer-based application related to natural language: machine translation (MT). It subsequently inspired many projects, notably the Georgetown experiment (Dostert, 1955), a joint project between IBM and Georgetown University that successfully demonstrated the machine

translation of more than 60 Russian sentences into English. The researchers accomplished this feat by using hand-coded language rules, but the system failed to scale up to general translation. In fact, early work in MT was very simple: most systems used dictionary-lookup of appropriate words for translation and reordered the words after translation to fit the word-order rules of the target language. This obviously produced very poor results, as the lexical ambiguity inherent in natural language was not taken into account. The researchers then progressively realized that the task was a lot harder than anticipated, and they needed a more adequate theory of language. It took until 1957 to introduce the idea of generative grammar (Chomsky, 1957), a rule based system of syntactic structures that brought insight into how mainstream linguistics could help machine translation.

Due to the development of the syntactic theory of language and parsing algorithms, the 1950s were flooded with over-enthusiasm. People believed that fully automatic high quality translation systems would be able to produce results indistinguishable from those of human translators, and that such systems would be in operation within a few years. Given the then-available linguistic knowledge and computer systems, this thought was completely unrealistic. In 1966, after more than a decade of research and millions of dollars spent, machine translations were still more expensive than manual human translations, and there were no computers that came anywhere near being able to carry on a basic conversation. That year, the ALPAC[1] released a report (Pierce et al., 1966) that concluded that MT was not immediately achievable and recommended the research community to stop funding it. This had the effect of substantially slowing down not only MT research, but also most work in other applications of NLP.

Despite this significant slowdown, some interesting developments were born during the years following the ALPAC report, both in theoretical issues and in construction of prototype systems. Theoretical work in the late 1960s and early 1970s mainly focused on how to represent meaning. Researchers developed new theories of grammar that were computationally tractable for the first time, particularly after the introduction of transformational generative grammars (Chomsky, 1965), which were criticised for being too syntactically oriented and not lending themselves easily to computational implementation. As a result, many new theories appeared to explain syntactic anomalies and provide semantic representations, such as case grammar (Fillmore, 1968), semantic networks (Collins et al., 1969), augmented transition networks (Woods, 1970), and conceptual dependency theory (Schank, 1972). Alongside theoretical development, this period of time also saw the birth of many interesting prototype systems. ELIZA (Weizenbaum, 1966) was built to replicate the conversation between a psychologist and a patient, simply by permuting or echoing the user input. SHRDLU (Winograd, 1971) was a simulated robot that used natural language to query and manipulate objects inside a very simple virtual micro-world consisting of a number of color blocks and pyramids. LUNAR (Woods et al., 1972) was developed as an interface system to a database that consisted of information about lunar rock samples using augmented transition network. Lastly, PARRY (Colby, 1974) attempted to simulate a person with paranoid schizophrenia based on concepts, conceptualizations, and beliefs.

The 1970s brought new ideas into NLP, such as building conceptual ontologies which structured real-world information into computer-understandable data. Examples are MARGIE (Schank and Abelson, 1975), TaleSpin (Meehan, 1976), QUALM (Lehnert, 1977), SAM (Cullingford, 1978), PAM (Schank and Wilensky, 1978) and Politics (Carbonell, 1979).

In the 1980s, many significant problems in NLP were addressed using symbolic approaches

[1]The Automatic Language Processing Advisory Committee of the National Academy of Science (ALPAC) was a committee of seven scientists led by John R. Pierce, established in 1964 by the United States government in order to evaluate the progress in computational linguistics in general and machine translation in particular.

(Charniak, 1983; Dyer, 1983; Riesbeck and Martin, 1986; Grosz et al., 1987; Hirst, 1987), i.e., complex hard-coded rules and grammars to parse language. Practically, text was segmented into meaningless tokens (words and punctuation). Representations were then manually created by assigning meanings to these tokens and their mutual relationships through well-understood knowledge representation schemes and associated algorithms. Those representations were eventually used to perform deep analysis of linguistic phenomena.

It wasn't until the late 1980s and early 1990s that statistical models came as a revolution in NLP (Bahl et al., 1989; Brill et al., 1990; Chitrao and Grishman, 1990; Brown et al., 1991), replacing most natural language processing systems based on complex sets of hand-written rules. This progress was the result of both the steady increase of computational power, and the shift to machine learning algorithms. While some of the earliest-used machine learning algorithms, such as decision trees (Tanaka, 1994; Allmuallim et al., 1994), produced systems similar in performance to the old school handwritten rules, statistical models broke through the complexity barrier of hand-coded rules by creating them through automatic learning, which led research to increasingly focus on these models. At the time, these statistical models were capable of making soft, probabilistic decisions.

2.1.2 NLP in the Deep Learning Era

Figure 2.2: The big stages of NLP in the deep learning era.

Starting in the 2000s, neural networks begin to be used for language modeling, a task which aims at predicting the next word in a text given the previous words. In 2003, Bengio et al. proposed the first neural language model, that consists of a one-hidden layer feed-forward neural network. They were also one of the first to introduce what is now referred as *word embedding*, a real-valued word feature vector in $\mathbb{R}^d$ (see Section 2.2). More precisely, their model took as input vector representations of the n previous words, which were looked up in a table learned together with the model. The vectors were fed into a hidden layer, whose output was then provided to a softmax layer that predicted the next word of the sequence. Although classic feed-forward neural networks have been progressively replaced with recurrent neural networks (Mikolov et al., 2010) and long short-term memory networks (Graves, 2013) for language modeling, they remain in some settings competitive with recurrent architectures, the latter being impacted by "catastrophic forgetting" (Daniluk et al., 2017). Furthermore, the general building blocks of Bengio et al.'s network are still found in most neural language and word embedding models nowadays.

In 2008, Collobert and Weston applied multi-task learning, a sub-field of machine learning in which multiple learning tasks are solved at the same time, to neural networks for NLP. They used a single convolutional neural network architecture (CNN; LeCun et al., 1999) that, given a sentence, was able to output many language processing predictions such as part-of-speech tags,

named entity tags and semantic roles. The entire network was trained jointly on all the tasks using weight-sharing of the look-up tables, which enabled the different models to collaborate and share general low-level information in the word embedding matrix. As models are being increasingly evaluated on multiple tasks to gauge their generalization ability, multi-task learning has gained in importance and is now used across a wide range of NLP tasks. Also, their paper turned out to be a discovery that went beyond multi-task learning. It spearheaded ideas such as pre-training word embeddings and using CNNs for text, that have only been widely adopted in the last years.

In 2013, Mikolov et al. introduced arguably the most popular word embedding model: Word2Vec. Although dense vector representations of words have been used as early as 2003 (Bengio et al.), the main innovation proposed in their paper was an efficient improvement of the training procedure, by removing the hidden layer and approximating the objective. Together with the efficient model implementation, these simple changes enabled large-scale training of word embeddings on huge corpora of unstructured text. Later that year (2013b), they improved the Word2Vec model by employing additional strategies to enhance training speed and accuracy. While these embeddings are not different conceptually than the ones learned with a feed-forward neural network, training on a very large corpus enables them to capture certain relationships between words such as gender, verb tense, and country-capital relations, which initiated a lot of interest in word embeddings as well as in the origin of these linear relationships (Mimno and Thompson, 2017; Arora et al., 2018; Antoniak and Mimno, 2018; Wendlandt et al., 2018). But what made word embeddings a mainstay in current NLP was the evidence that using pre-trained embeddings as initialization improved performance across a wide range of downstream tasks. Since then, a lot of work has gone into exploring different facets of word embeddings (as indicated by the staggering number of citations of the original paper).[2] Despite many more recent developments, Word2Vec is still a popular choice and widely used today.

The year 2013 also marked the adoption of neural network models in NLP, in particular three well-defined types of neural networks: recurrent neural networks (RNNs; Elman, 1990), convolutional neural networks (CNNs), and recursive neural networks (Socher et al., 2013). Because of their architecture, RNNs became popular for dealing with the dynamic input sequences ubiquitous in NLP. But Vanilla RNNs were quickly replaced with the classic long-short term memory networks (LSTMs; Hochreiter and Schmidhuber, 1997), as they proved to be more resilient to the vanishing and exploding gradient problem. At the same time, convolutional neural networks, that were then beginning to be widely adopted by the computer vision community, started to get applied to natural language (Kalchbrenner et al., 2014; Kim, 2014). The advantage of using CNNs for dealing with text sequences is that they are more parallelizable than RNNs, as the state at every time step only depends on the local context (via the convolution operation) rather than all past states as in the RNNs. Finally, recursive neural networks were inspired by the principle that human language is inherently hierarchical: words are composed into higher-order sentences, which can themselves be recursively combined according to a set of production rules. Based on this linguistic perspective, recursive neural networks treated sentences as trees rather than as a sequences. Some research (Tai et al., 2015) also extended RNNs and LSTMs to work with hierarchical structures.

In 2014, Sutskever et al. proposed sequence-to-sequence learning, a general end-to-end approach for mapping one sequence to another using a neural network. In their method, an encoder neural network processes a sentence symbol by symbol, and compresses it into a vector representation. Then, a decoder neural network predicts the output sequence symbol by symbol based

[2] At the time of writing this thesis, "*Distributed representations of words and phrases and their compositionality*" has 19,071 citations at its credit.

on the encoder state and the previously predicted symbols that are taken as input at every step. Encoders and decoders for sequences are typically based on RNNs, but other architectures have also emerged. Recent models include deep-LSTMs (Wu et al., 2016), convolutional encoders (Kalchbrenner et al., 2016; Gehring et al., 2017), the Transformer (Vaswani et al., 2017), and a combination of an LSTM and a Transformer (Chen et al., 2018). Machine translation turned out to be the perfect application for sequence-to-sequence learning. The progress was so significant that Google announced in 2016 that it was officially replacing its monolithic phrase-based machine translation models in Google Translate with a neural sequence-to-sequence model.

In 2015, Bahdanau et al. introduced the principle of *attention*, which is one of the core innovations in neural machine translation (NMT) and the key idea that enabled NMT models to outperform classic sentence-based MT systems. It basically alleviates the main bottleneck of sequence-to-sequence learning, which is its requirement to compress the entire content of the source sequence into a fixed-size vector. Indeed, attention allows the decoder to look back at the source sequence hidden states, that are then combined through a weighted average and provided as additional input to the decoder. Attention is potentially useful for any task that requires making decisions based on certain parts of the input. For now, it has been applied to constituency parsing (Vinyals et al., 2015), reading comprehension (Hermann et al., 2015), and one-shot learning (Vinyals et al., 2016). More recently, a new form of attention has appeared, called *self-attention*, being at the core of the Transformer architecture. In short, it is used to look at the surrounding words in a sentence or paragraph to obtain more contextually sensitive word representations (see Section 3.1 for detailed explanation).

The latest major innovation in the world of NLP is undoubtedly large pre-trained language models. While first proposed in 2015 (Dai and Le), only recently were they shown to give a large improvement over the state-of-the-art methods across a diverse range of tasks. Pre-trained language model embeddings can be used as features in a target model (Peters et al., 2018), or a pre-trained language model can be fine-tuned on target task data (Devlin et al., 2018; Howard and Ruder, 2018; Radford et al., 2019; Yang et al., 2019), which have shown to enable efficient learning with significantly less data. The main advantage of these pre-trained language models comes from their ability to learn word representations from large unannotated text corpora, which is particularly beneficial for low-resource languages where labelled data is scarce.

2.2 Word Embedding

By definition, a *word embedding* is a dense, fixed-length, real-valued vector representation of a word. Consequently, a word embedding model $\mathcal{W} \rightarrow \mathbb{R}^n$ is a parameterized function that maps words $w \in \mathcal{W}$ to high-dimensional real vectors.

In the last decade, word embeddings have established themselves as a core element of many NLP systems. Indeed, NLP methods deal with natural language, which often appears in the form of text. This text is itself composed of smaller units like words and characters, which are not directly understandable by computers in any human sense. As a result, word embeddings are needed to numerically represent textual input which can be read, understood and processed by computer programs. This need was emphasized with the recent explosion of deep learning, which proved to solve a huge amount of problems in various fields. NLP researchers who wanted to use promising deep models on text data had to think at a numerical way to represent words. This resulted in an extensive research for best representing textual data such that the representations capture both semantic and syntactic meanings of words.

In practice, there are several ways to represent a word by a vector. The simplest one is probably *one-hot encoding*. Given a vocabulary of N words, this method consists in assigning an integer index $i \in \{1, ..., N\}$ to each word. With this word-to-integer mapping, a word is

then represented as a N-dimensional sparse vector mostly composed of zeros, except for a single entry at the position corresponding to the word's index in the vocabulary that takes the value 1. These one-hot vectors are a quick and easy way to represent words numerically, but present two main problems. First, this approach suffers from an obvious feature size downside, as the vector size increases with the size of the vocabulary. More features mean more parameters to estimate, which in turn require exponentially more data for their estimation as well as additional computational power. Second, one-hot encoding does not take similarities between word into accounts, as shown on the left-hand side of Figure 2.3. Ideally, we would like an embedding to linguistically capture meaningful relationships between words, as shown on the right-hand side of Figure 2.3. For these reasons, one-hot encoding is very little used directly as a word vector representation but rather as an input to more elaborated embedding methods.

Figure 2.3: Two different approaches for encoding words into vector spaces. One-hot encoding on the left-hand side; semantically-meaningful word embedding on the right-hand side.

To address the similarity issue encountered with one-hot encoding, NLP researchers turned to a very powerful idea introduced by the American linguist Zellig S. Harris: the distributional hypothesis (Harris, 1954). It states that words that occur in the same contexts tend to have similar meanings. This theory was further popularized by the English linguist John R. Firth who famously said: "You shall know a word by the company it keeps" (Firth, 1957). Commonly used word embedding methods all rely on this assumption in some way. However, the techniques used to implement it differ. Basically, word embedding models fall into two main categories. On one hand, *count-based* models use corpus-wide statistics such as word counts and frequencies to build word representations. On the other hand, *prediction-based* models learn embeddings by maximizing their predictive ability, i.e by trying to predict a word given its context or inversely. Very recently, a third category has appeared: *deep contextual* models, which have the particularity of using local context to create a word representation. This section briefly introduces some of the more popular word embedding methods for the three categories.

2.2.1 Count-based Models

Count-based models are based on a co-occurrence matrix. This matrix is built by looping over a massive dataset of textual documents in order to accumulate word co-occurrence counts, i.e., the number of times two or more words occur together in the dataset. In practice, there exists two types of co-occurrence: *word-document* and *window-based*. A *word-document* matrix $\boldsymbol{X} \in \mathbb{N}^{N \times D}$ (N is the number of words in the vocabulary and D is the total number of documents in the dataset) is such that the entry $\boldsymbol{X}_{ij}$ corresponds to the number of times word i appears in document j. The *window-based* matrix $\boldsymbol{X'} \in \mathbb{N}^{N \times N}$, however, is such that the entry $\boldsymbol{X'}_{ij}$ represents the number of times the word i appears in a specified sized window around another word j. An example of such matrix is given in Figure 2.4. Once the co-occurrence matrix

has been computed, its dimensionality is typically reduced using Singular Value Decomposition (SVD) such that $\boldsymbol{X}$ (respectively $\boldsymbol{X'}$) is factorized into $\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal matrices. Finally, the rows of $\boldsymbol{U}$ are used as the word embeddings for all words in the vocabulary.

$$
X' = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array}
\begin{array}{c}
\begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\
\begin{bmatrix}
0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{bmatrix}
\end{array}
$$

Figure 2.4: Example of a window-based co-occurrence matrix computed with a window size of 3 (one word on either side of the input word, in addition to the input word itself), over the following corpus: "I enjoy flying.", "I like NLP." and "I like deep learning.".

Well-known count-based models include Latent Semantic Analysis (LSA; Landauer and Dumais, 1997), Hyperspace Analog to Language (HAL; Lund and Burgess, 1996), Correlated Occurrence Analogue to Lexical Semantics (COALS; Rohde et al., 2004) and Hellinger-PCA (Lebret and Collobert, 2013). Although these methods effectively leverage global statistical information, they are primarily used to capture word similarities and do poorly on tasks such as word analogy, indicating a sub-optimal vector space structure.

2.2.2 Prediction-based Models

Neural Network Language Model

The *neural network language model* (NNLM; Bengio et al., 2003) jointly learns a word vector representation and a statistical language model with a feed-forward neural network that contains a linear projection layer and a non-linear hidden layer. Given a sequence $w_1 ... w_n$ of n words $w_t \in V$, where n is the window size and V the vocabulary, the objective is to learn a function $f(w_t, \ldots, w_{t-n+1}) = p(w_t | w_{t-1}, \ldots, w_{t-n+1})$. In their model, this function is a composition of two mappings C and g such that

$$f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1})), \tag{2.1}$$

where

- the mapping C represents the *distributed feature vectors* associated with each word in the vocabulary. In practice, it appears as a $|V| \times m$ matrix of free parameters, and maps a word index to its m-dimensional feature representation $C(i) : i \to \mathbb{R}^m, i = \{1, \cdots, |V|\}$.
- the mapping g takes an input sequence of feature vectors for words in context $C(w_{t-n+1})$, ..., $C(w_{t-1})$ and maps it to a conditional probability distribution over words in V for the next word w_t. In practice, it outputs a vector whose i-th element estimates the probability $p(w_t = i | w_{t-1}, \cdots w_{t-n+1})$ thanks to a softmax layer, as shown in Figure 2.5.

Training is achieved by looking for $\boldsymbol{\theta}$ that maximizes the training corpus log-likelihood:

$$\begin{aligned} \mathcal{L} &= \frac{1}{T} \sum_t \log f(w_t, w_{t-1}, \ldots, w_{t-n+1}; \boldsymbol{\theta}) \\ &= \frac{1}{T} \sum_t \log p(w_t | w_{t-1}, \ldots, w_{t-n+1}). \end{aligned} \tag{2.2}$$

Figure 2.5: Neural network language model (NNLM) architecture (Bengio et al., 2003). $f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1}))$, where g is the neural network and $C(i)$ is the i-th word feature vector.

The main drawback of such language models is undoubtedly the final softmax layer, as the cost of computing the softmax is proportional to the number of words in V, which is typically on the order of hundreds of thousands or millions. Therefore, NNLM models are very computationally expensive.

Continuous Bag-of-Words Model

While a language model is only able to look at the past words for its predictions (as it is evaluated on its ability to predict each next word given a sequence of previous words), a model that just aims to generate accurate word embeddings can bypass this restriction. The *continuous bag-of-words* model (CBOW; Mikolov et al., 2013b) uses both the n words before and after the target word w_t for its prediction (hence the name "continuous", as it uses continuous representations whose order is of no importance). Hence, the training objective is slightly different than the one of the NNLM, presented in Equation (2.2). Instead of feeding the n previous words into the model, it receives a window of n words around the target word w_t at each time step t such that the loss becomes

$$\mathcal{L} = \frac{1}{T} \sum_t \log p\left(w_t | w_{t-n}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+n}\right). \tag{2.3}$$

In addition, the model architecture differs from the feed-forward NNLM in the sense that the non-linear hidden layer has been removed, reducing the computational cost. The CBOW architecture is shown in Figure 2.6a.

In practice, the weights between the input layer and the hidden layer can be represented by a matrix $\boldsymbol{W} \in \mathbb{R}^{|V| \times d}$, where V is the vocabulary and d the hidden layer size. Each row of $\boldsymbol{W}$ is the d-dimensional vector representation of the associated word from the input layer. Given a context word, it is first represented as a one-hot encoded vector $\boldsymbol{x}$, where only the k-th unit out of $|V|$ is 1 while all other units are 0. The vector $\boldsymbol{x}$ is then multiplied against $\boldsymbol{W}$ so that

$$\boldsymbol{h} = \boldsymbol{W}^\top \boldsymbol{x} = \boldsymbol{W}_{(k,\cdot)}^\top = \boldsymbol{v}_{w_I}{}^\top, \tag{2.4}$$

which essentially comes down to copying the k-th row of $\boldsymbol{W}$ to $\boldsymbol{h}$. In a multi-word context setting, the CBOW model takes the average of the input vector representations such that

$$\begin{aligned} \boldsymbol{h} &= \frac{1}{n}\boldsymbol{W}^\top\left(\boldsymbol{x}_1 + \boldsymbol{x}_2 + \cdots + \boldsymbol{x}_n\right) \\ &= \frac{1}{n}\left(\boldsymbol{v}_{w_1} + \boldsymbol{v}_{w_2} + \cdots + \boldsymbol{v}_{w_n}\right)^\top, \end{aligned} \tag{2.5}$$

where n is the number of words in the context. Similarly, the weights between the hidden layer and the output layer are represented by another matrix $\boldsymbol{W'} \in \mathbb{R}^{d\times|V|}$. Using these weights, a score s_j is computed for each word in the vocabulary such that

$$s_j = \boldsymbol{v'}_{w_j}{}^\top \boldsymbol{h}, \tag{2.6}$$

where $\boldsymbol{v'}_{w_j}$ is the j-th column of the matrix $\boldsymbol{W'}$. Then, the softmax operation is used to obtain a posterior distribution of words, expressed as

$$p\left(w_j|w_I\right) = y_j = \frac{\exp\left(s_j\right)}{\sum_{j'=1}^{|V|}\exp\left(s_{j'}\right)}, \tag{2.7}$$

where y_j is the output of the j-th unit in the output layer. Once the parameters of both matrices $\boldsymbol{W}$ and $\boldsymbol{W'}$ have been learned, they can either be used directly or averaged together to obtain the final word embedding matrix. The CBOW model is at the core of the first implementation of arguably the most popular word embedding method: Word2Vec (Mikolov et al.; 2013a, 2013b).

Skip-Gram Model

The *skip-gram* model (SG; Mikolov et al., 2013b) is very similar to CBOW, but instead of predicting the current word based on the context, it predicts the surrounding context words given a center word w_t. Hence, the skip-gram objective thus sums the log probabilities of the surrounding n words to the left and to the right of the target word w_t and maximizes the average log probability, written as

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{\top}\sum_{-c\leq j\leq c, j\neq 0}\log p\left(w_{t+j}|w_t\right). \tag{2.8}$$

In skip-gram, the transition from input layer to hidden layer works similarly to CBOW in a one-word context, as presented in Equation (2.4). It means $\boldsymbol{h}$ is simply copying (and transposing) a row of the "input → hidden" matrix $\boldsymbol{W}$, associated with the unique input word w_I. The difference comes in the output layer, which instead of outputting one multinomial distribution, outputs n multinomial distributions. The architecture of the skip-gram model is shown in Figure 2.6b.

The skip-gram model is at the basis of the second implementation of the famous Word2Vec method (Mikolov et al.; 2013a, 2013b), as well as the well-known FastText (Bojanowski et al., 2017). The particularity of FastText is that, instead of learning representations for words directly, it represents each word as an n-gram[3] of characters and learns a representation for each character n-gram, so that the overall word embedding is a sum of these n-gram representations. The main advantage of this approach is that it can handle rare and out-of-corpus words, what Word2Vec fails to do.

In addition to the CBOW and SG models, Mikolov et al. came up with additional strategies to enhance speed and accuracy (2013b), which were as revolutionary as the models themselves.

[3]In the fields of computational linguistics, an n-gram is a contiguous sequence of n items from a given sample of text or speech, where the items can be phonemes, syllables, letters or subwords according to the application (e.g., the word "artificial" with n=3 might give the following n-grams: art, rti, tif, ifi, fic, ici, ial).

These methods mainly focus on improving the final probability distribution computation, which was initially inefficiently performed by a softmax function as presented in Equation (2.7). Alternatives include negative sampling and hierarchical softmax. In short, negative sampling is a method that maximizes the log-likelihood of the softmax model by only summing over a smaller subset of m ($m < |V|$) words. Regarding hierarchical softmax, it uses a binary tree representation of the output layer where the words are leaves and every node represents the relative probabilities of its child nodes. These two methods greatly improve the performance of the initial CBOW and SG models.

(a) Continuous bag-of-words model.

(b) Skip-gram model.

Figure 2.6: *Continuous bag-of-words* (CBOW) and *skip-gram* (SG) architectures (Rong, 2014). The CBOW architecture predicts the current word based on the context, and SG predicts surrounding words given the current word.

2.2.3 Deep Contextual Models

Until then, all word embeddings techniques were missing a crucial element for fully capturing semantic and syntactic meanings of words: *local* context. Indeed, widely used word representation methods such as Word2Vec or FastText are all context-independent. These methods actually learn to capture the *general* (i.e., the most common) context of words in their representations, by observing that a given word often occurs within one context or another, but they are by no means able to handle polysemy,[4] as a single word with multiple meanings is always mapped to one unique vector. Hence, such techniques usually work incredibly well with words that represent a unique concept (e.g., an animal or a country), but miss to capture the real meaning of a word which unavoidably depends on its local context.

To address this problem, researchers have taken an increasing interest to *deep contextual* word representations. The idea is simple: a token is assigned a representation that is a function of the entire input sentence. Early work focused on learning context-dependent representations included Context2Vec (Melamud et al., 2016), CoVe (McCann et al., 2017), ELMO (Peters et al., 2018) and ULMFiT (Howard and Ruder, 2018), that all rely on a bidirectional LSTM to encode

[4]Polysemous words are words with two or more meanings (e.g., "run" is currently the largest polysemous entry in the Oxford English Dictionary with 645 meanings).

the context. More recently, the introduction of the Transformer architecture (Vaswani et al., 2017) resulted in a series of powerful pre-trained language representation models such as BERT (Devlin et al., 2018), XLNet (Yang et al., 2019) and ERNIE (Zhang et al., 2019a), that proved their effectiveness in a wide variety of language tasks.

2.3 Domain-specific BERT-based Models

Related to this thesis, the idea of investigating how state-of-the-art language representation models can be adapted to a specific domain has recently generated interest in a few areas.

For example, Huang et al. (2019) pre-trained a BERT model on clinical notes. In one of their experiments, they compared their novel model, ClinicalBERT, to popular word embedding models, using a clinical word similarity task. They found that ClinicalBERT exhibits higher correlation with human evaluation than BERT on some medical concepts. In addition, they showed that ClinicalBERT outperforms BERT on clinical language modeling tasks. However, no comparison was performed between ClinicalBERT and BERT on domain-specific downstream NLP tasks. Instead, they fine-tuned their novel model on clinical predictions, and compared it with two baselines, including a bag-of-words model and an LSTM model with Word2Vec embeddings as inputs. Their results showed that ClinicalBERT largely outperforms these baselines for clinical predictive tasks. In their work, the parameters of ClinicalBERT were initialized with the released BERT parameters.

Similarly, Lee et al. (2020) introduced BioBERT, a BERT model pre-trained on a very large biomedical corpus (about 5.5 times larger than the corpus used to pre-train BERT). By further fine-tuning their model on three representative biomedical text mining tasks, including named entity recognition (NER), relation extraction (RE) and question answering (QA), they showed that BioBERT largely outperforms BERT and previous state-of-the-art models on these tasks for a variety of biomedical datasets. Here again, BioBERT was initialized with the weights from the original BERT.

Lastly, Beltagy et al. (2019) released SciBERT, a BERT model pre-trained on a large corpus of scientific publications (about the same size as the corpus on which BERT was pre-trained). In this work, they particularly investigated the effect of fine-tuning the pre-trained model on downstream tasks versus using task-specific architectures atop frozen pre-trained embeddings. By evaluating both methods on text classification, sequence labeling and dependency parsing, they found that the fine-tuning approach led to much better results than the feature-based approach. Interestingly, they even showed that BERT with fine-tuning slightly outperforms (or performs similarly to) another task-specific model using frozen SciBERT embeddings. In general, fine-tuned SciBERT largely outperforms fine-tuned BERT on the three downstream tasks mentioned earlier. Another key contribution of their work is their study on the differences between pre-training BERT from scratch with an in-domain vocabulary or initializing the model parameters with BERT's weights. Eventually, they found that the optimal hyperparameters for SciBERT pre-trained from scratch often coincided with those of SciBERT initialized with BERT's parameters. They suspect that while an in-domain vocabulary might be helpful, SciBERT benefits most from the scientific corpus pre-training.

Chapter 3

BERT

BERT (Devlin et al., 2018), which stands for Bidirectional Encoder Representation from Transformer, is a deep contextual language representation model introduced by Google AI researchers. It is designed to pre-train deep bidirectional representations of words from unlabeled text by jointly conditioning on both the left and right contexts in all its layers. As a result, the pre-trained model can be fine-tuned with just one additional output layer to create very performing models for a wide range of NLP tasks, such as text classification, as shown in Figure 3.1.

This chapter introduces BERT and its detailed implementation. In particular, Section 3.1 explains the *self-attention* mechanism adopted in the model. Then, Section 3.2 describes the model itself, its architecture, input representations and parameters. Next, Section 3.3 details the pre-training procedure of the model. Finally, Section 3.4 explains the different approaches for applying the model to downstream tasks.

Figure 3.1: Learning steps of BERT. (1) Pre-training: self-supervised training on large amounts of text (books & Wikipedia). (2) Fine-tuning: supervised training on a specific task with a labeled dataset.

3.1 Self-attention

Self-attention is a particular form of *attention* (Bahdanau et al., 2014) that was first introduced with the Transformer model (Vaswani et al., 2017). Simply put, it is an attention mechanism

relating different positions of a single sequence in order to compute a contextual representation for each term of that sequence, as illustrated in Figure 3.2.

Figure 3.2: Visualization of the self-attention mechanism. This example shows the attention brought to the word "it" from all the words in the sentence (in one of the heads of the last attention-layer in BERT).[1]

Formally, *self-attention* can be described as mapping a query and a set of key-value pairs to an output, where the queries, keys, values and outputs are all vectors. More precisely, given an input sequence of size N, the self-attention mechanism performs the following steps for each term i ($i = 1, ..., N$) in the sequence:

1. Compute a query vector $\boldsymbol{q}_i$ and a key vector $\boldsymbol{k}_i$ both of dimension d_k, as well as a value vector $\boldsymbol{v}_i$ of dimension d_v. These vectors are obtained by multiplying an initial embedding $\boldsymbol{x}_i \in \mathbb{R}^{d_{\text{model}}}$ of the term i with three weight matrices $\boldsymbol{W}^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\boldsymbol{W}^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $\boldsymbol{W}^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ learned during the training process:

$$\begin{aligned} \boldsymbol{q}_i &= \boldsymbol{x}_i \boldsymbol{W}^Q \\ \boldsymbol{k}_i &= \boldsymbol{x}_i \boldsymbol{W}^K \\ \boldsymbol{v}_i &= \boldsymbol{x}_i \boldsymbol{W}^V. \end{aligned} \tag{3.1}$$

2. Score the term i against all the other terms in the sequence by taking the dot product of its query vector $\boldsymbol{q}_i$ with all the key vectors $\boldsymbol{k}_j$ of the sequence:

$$s_{ij} = \boldsymbol{q}_i \boldsymbol{k}_j, \qquad \forall j = 1, ..., N. \tag{3.2}$$

3. Divide the scores of the term i by the square root of the key vector dimension d_k:

$$s'_{ij} = \frac{s_{ij}}{\sqrt{d_k}}, \qquad \forall j = 1, ..., N. \tag{3.3}$$

[1]Example created from https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb

4. Pass the new scores of the term i through a softmax operation to normalize them:

$$s''_{ij} = \frac{e^{s'_{ij}}}{\sum_{j=1}^{N} e^{s'_{ij}}}, \qquad \forall j = 1, ..., N. \tag{3.4}$$

5. Multiply each value vector $\boldsymbol{v}_j$ with their corresponding normalized score:

$$\boldsymbol{v}'_{ij} = s''_{ij}\boldsymbol{v}_j, \qquad \forall j = 1, ..., N. \tag{3.5}$$

6. Sum up the weighted value vectors as the final output of the self-attention calculation:

$$\boldsymbol{z}_i = \sum_{j=1}^{N} \boldsymbol{v}'_{ij}. \tag{3.6}$$

Note that the third step of the computation aims at solving a problem suspected by Vaswani et al. (2017), who claim that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. The division operation by d_k counteracts this effect.

Figure 3.3 gives an example of the self-attention mechanism in vector form. The left-hand side of the figure describes the step 1 of the self-attention computation, while the right-hand shows the steps 2 to 6.

(a) Computation of the query, key and value vectors $\boldsymbol{q}_i$, $\boldsymbol{k}_i$ and $\boldsymbol{v}_i$, respectively.

(b) Computation of the self-attention output vectors $\boldsymbol{z}_i$.

Figure 3.3: Illustration of the self-attention mechanism computed in vector form (Jay Alammar, 2018).

In practice, the self-attention function is computed on a set of queries simultaneously, packed together into a matrix $\boldsymbol{Q} \in \mathbb{R}^{N \times d_k}$. The keys and values are also packed together into respective matrices $\boldsymbol{K} \in \mathbb{R}^{N \times d_k}$ and $\boldsymbol{V} \in \mathbb{R}^{N \times d_v}$, as shown in Figure 3.4a. That way, the output matrix is computed as follows:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}. \tag{3.7}$$

Figure 3.4b illustrates the latter computation.

(a) Computation of the query, key and value matrices $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$, respectively.

(b) Computation of the self-attention output matrix $\boldsymbol{Z}$.

Figure 3.4: Illustration of the self-attention mechanism computed in matrix form (Jay Alammar, 2018).

3.1.1 Multi-head Attention

Instead of performing a single self-attention operation with d_{model}-dimensional keys, values and queries, the Transformer-based models actually go a step further by using a mechanism called "multi-head" attention. With multi-head attention, the queries $\boldsymbol{Q}$, keys $\boldsymbol{K}$ and values $\boldsymbol{V}$ are linearly projected h times with different learned linear projections $\boldsymbol{P}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\boldsymbol{P}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\boldsymbol{P}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ $(i = 1, ..., h)$. This allows to jointly attend to information from different representation subspaces at different positions. Hence, the self-attention function is performed on h different projections of the query, key and value matrices in parallel. This yields h different output matrices $\boldsymbol{Z}_i \in \mathbb{R}^{N \times d_{\text{model}}}$ called "attention heads", as depicted in Figure 3.5a. The attention heads are then concatenated and projected into another representation subspace with a matrix $\boldsymbol{W}^O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$, resulting in the final multi-head attention output matrix $\boldsymbol{Z} \in \mathbb{R}^{N \times d_{\text{model}}}$, as shown in Figure 3.5b. In brief, the final output is computed as follows:

$$\begin{aligned} \text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) &= \text{Concat}\left(\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_\text{h}\right) \boldsymbol{W}^O \\ \text{where } \boldsymbol{Z}_\text{i} &= \text{Attention}\left(\boldsymbol{Q}\boldsymbol{P}_i^Q, \boldsymbol{K}\boldsymbol{P}_i^K, \boldsymbol{V}\boldsymbol{P}_i^V\right), \quad i = 1, ..., h. \end{aligned} \tag{3.8}$$

Note that in multi-head attention, it is usual that $d_k = d_v = d_{\text{model}}/h$.

(a) Computation of the attention heads $\boldsymbol{Z}_i$ $(i = 1, ..., h)$.

(b) Computation of the final multi-head attention output $\boldsymbol{Z}$.

Figure 3.5: Illustration of the multi-head attention mechanism (Jay Alammar, 2018). This example shows $h = 8$ different attention heads, as in the original Transformer implementation. Note that BERT actually has 12 or 16 attention heads depending on its version.

3.2 Model

3.2.1 Architecture

BERT's architecture is a multi-layer bidirectional Transformer encoder (Vaswani et al., 2017). In other words, BERT is composed of a stack of L identical Transformer encoder layers. Each encoder layer contains two types of sublayer. The first is a multi-head self-attention mechanism, which helps look at other words in the sequence while encoding a specific word. The second is a simple, position-wise fully connected feed-forward network (FFN), which is applied to each position separately and identically, and consists of two linear transformations $\left(\boldsymbol{W}_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}, \boldsymbol{b}_1 \in \mathbb{R}^{d_{\text{ff}}}\right)$, $\left(\boldsymbol{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}, \boldsymbol{b}_2 \in \mathbb{R}^{d_{\text{model}}}\right)$ such that

$$\text{FFN}(\boldsymbol{x}) = \max(0, \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2. \tag{3.9}$$

The dimensionality of input and output is d_{model} and the inner-layer has dimensionality $d_{\text{ff}} = 4d_{\text{model}}$. The feed-forward network also uses a GELU activation (Hendrycks and Gimpel, 2016), defined as

$$\text{GELU}(x) = 0.5x\left(1 + \tanh\left(\sqrt{2/\pi}\left(x + 0.044715x^3\right)\right)\right), \tag{3.10}$$

which was shown to work better than the standard ReLU (Nair and Hinton, 2010) within a Transformer encoder. In addition, an encoder layer employs a residual connection (He et al., 2016) around each of the two sublayers, followed by a layer normalization (Ba et al., 2016) such that the output of each sublayer is

$$\text{LayerNorm}(\boldsymbol{x} + \text{Sublayer}(\boldsymbol{x})), \tag{3.11}$$

where $\text{Sublayer}(\boldsymbol{x})$ represents the function implemented by the sublayer itself. To facilitate these residual connections, all sublayers in the model produce outputs of the same dimension d_{model}. The architecture of a single encoder is shown in Figure 3.6, and an additional 3D visualization of BERT's structure is shown in Figure A.1. Note that while the linear transformations are the same across different positions within the same sublayer, BERT uses different parameters from layer to layer.

Figure 3.6: Architecture of the Transformer Encoder (Jay Alammar, 2018).

As shown in Figure 3.7, BERT comes in two versions:

- BERT-base: L=12, d_{model}=768, h=12, d_{ff}=3072 (110M total parameters).
- BERT-large: L=24, d_{model}=1024, h=16, d_{ff}=4096 (340M total parameters).

Here, L denotes the number of layers, d_{model} the dimensionality of input and output of each layer, h the number of attentions heads in a self-attention sublayer, and d_{ff} the number of hidden units in a feed-forward sublayer.

Figure 3.7: High-level illustration of BERT architecture in its two versions (Jay Alammar, 2019).

3.2.2 Input Representations

Given a sequence of words as inputs (limited to 512 tokens), BERT performs a first transformation of these words in order to obtain numerical input representations to pass to the model. In practice, these input representations are constructed by summing three different types of embedding: *token*, *segment* and *positional* embeddings. A visualization of this construction can be seen in Figure 3.8.

Figure 3.8: BERT input representation (Devlin et al., 2018). For each token in an input sequence, its input representation is the sum of its *token*, *segment* and *positional* embeddings.

Token Embedding

Given a word in the input sequence, BERT uses WordPiece embedding (Wu et al., 2016) to tokenize it. Basically, WordPiece is a model that creates a fixed-size vocabulary of individual characters, subwords and words that best fits a given language corpus. To tokenize a word under this model, the tokenizer first checks if the whole word is in the vocabulary. If not, it tries to break it into the largest possible subwords contained in the vocabulary, and as a last resort will decompose the word into individual characters. Once the word has been processed into one or multiple WordPiece tokens, the vocabulary ids of these tokens are used to retrieve the corresponding embeddings in the learned token embedding matrix, shown in Figure 3.9.

The vocabulary used by BERT contains the ∼30,000 most common words and subwords found in the English language, in addition to all English characters and three special tokens:

- `[CLS]`, which is used as a special classification token that appears at the beginning of every sequence. The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks. It is ignored in non-classification tasks.
- `[SEP]`, which is used as a delimiter when dealing with sentence pairs packed together into a single sequence. It also always ends the sequence.
- `[MASK]`, which is used for the masked language modeling (MLM) training objective, discussed in Section 3.3.1.

Segment Embedding

When dealing with sentence pairs, a learned *segment* embedding is added to every token indicating whether it belongs to sentence `A` or sentence `B`. Segment embeddings are similar to token embeddings with a simple vocabulary of size 2, as illustrated in Figure 3.9.

Positional Embedding

In order to inject some information about the relative or absolute position of the tokens in the input sequence, BERT uses *positional* embeddings, as in the original Transformer. These embeddings have the same dimension d_{model} as the token and segment embeddings so that they can easily be summed up. They are computed using sine and cosine functions of different frequencies:

$$\begin{aligned} \text{PE}_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \\ \text{PE}_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \end{aligned} \tag{3.12}$$

where pos is the position and i is the dimension. Hence, each dimension of the positional embedding corresponds to a sinusoid, and the wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. Vaswani et al. (2017) hypothesized that this function allows the model to easily learn the relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

Figure 3.9: Parameters of the input embedding layer.

3.2.3 Parameters

As mentioned earlier, BERT comes in two versions. The base version has about 110M parameters in total, while the large one has 340M parameters in total. These parameters include:

- the token, positional and segment embedding matrices of the input embedding layer, respectively defined as

$$\boldsymbol{W}^{TE} \in \mathbb{R}^{d_{\text{voc}} \times d_{\text{model}}}, \boldsymbol{W}^{PE} \in \mathbb{R}^{d_{\text{context}} \times d_{\text{model}}}, \boldsymbol{W}^{SE} \in \mathbb{R}^{2 \times d_{\text{model}}}.$$

- the query, key and value weight matrices of each self-attention sublayer, respectively defined as

$$\boldsymbol{W}^{Q} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \boldsymbol{W}^{K} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \boldsymbol{W}^{V} \in \mathbb{R}^{d_{\text{model}} \times d_v}.$$

- the h triplets of multi-head linear projections in each self-attention sublayer (where h refers to the number of attention heads), defined as

$$\left(\boldsymbol{P}_i^{Q} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \boldsymbol{P}_i^{K} \in \mathbb{R}^{d_{\text{model}} \times d_k}, \boldsymbol{P}_i^{V} \in \mathbb{R}^{d_{\text{model}} \times d_v}\right), \quad i = 1, ..., h.$$

- the multi-head output projection matrix in each self-attention sublayer, defined as

$$\boldsymbol{W}^{O} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}.$$

- the feed-forward network parameters in each feed-forward sublayer, defined as

$$\left(\boldsymbol{W}_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}, \boldsymbol{b}_1 \in \mathbb{R}^{d_{\text{ff}}}\right), \left(\boldsymbol{W}_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}, \boldsymbol{b}_2 \in \mathbb{R}^{d_{\text{model}}}\right).$$

- the residual connection parameters in each layer, defined as

$$\boldsymbol{W}^{R} \in \mathbb{R}^{d_{\text{model}}}.$$

In practice, we have

$$\begin{cases} d_{\text{model}} = 768 \text{ for BERT-base } (d_{\text{model}} = 1024 \text{ for BERT-large}); \\ d_{\text{voc}} = 30,522 \text{ for the } \textit{cased} \text{ vocabulary } (d_{\text{voc}} = 28,996 \text{ for the } \textit{uncased} \text{ vocabulary}); \\ d_{\text{context}} = 512; \\ d_{\text{ff}} = 4d_{\text{model}}; \\ d_k = d_v = \frac{d_{\text{model}}}{h}. \end{cases}$$

A visualization of all these parameters is given in Figure 3.9 and Figure 3.10. Additionally, a detailed breakdown of BERT parameters is given in Table A.1 for the base version, and in Table A.2 for the large one.

(a) Parameters of a Self-Attention sublayer.

(b) Parameters of a FFN sublayer.

Figure 3.10: Parameters of an Encoder Layer.

3.3 Pre-training Procedure

BERT is pre-trained simultaneously on two tasks: *masked language modeling* (MLM) and *next sentence prediction* (NSP). The training loss is simply the sum of the mean MLM likelihood and the mean NSP likelihood.

3.3.1 Masked Language Modeling

Unlike language modeling (LM), which aims at predicting the next word given the sequence of previous words, masked language modeling is the task of predicting a percentage of input tokens which are randomly masked.

The MLM training objective was chosen over the traditional LM objective because of the bidirectionality of BERT (i.e., BERT uses both left and right context in the sequence to predict the target word). For such models, standard conditional language modeling cannot be used as training objective, as the bidirectional conditioning would allow each word to indirectly "see itself", and the model could trivially predict the target word in a multi-layered context. Hence, BERT is trained using masked language modeling, also referred as a Cloze task in the literature (Taylor, 1953). As shown in Figure 3.11, the prediction is given by the final hidden vectors corresponding to the masked tokens, that are fed into an output softmax over the vocabulary, as in a standard LM.

Figure 3.11: Illustration of the *masked language modeling* (MLM) training objective (Jay Alammar, 2019).

Formally, given an input sequence $\boldsymbol{x} = [x_1, x_2, ..., x_N]$ of N tokens, MLM first selects a random set of k positions (integers between 1 and N) to mask out $\boldsymbol{m} = [m_1, ..., m_k]$. The tokens in the selected positions are then replaced with a `[MASK]` token, resulting in the masked input sequence $\boldsymbol{x}^{masked}$. BERT eventually learns to predict the original identities of the k masked-out tokens by computing an output word distribution $\hat{\boldsymbol{y}}^{(h)}$ $(h = 1, ..., k)$ for each one of them. More precisely, given the h-th masked word x_{m_h} from sequence $\boldsymbol{x}$, the MLM loss function is the cross-entropy between the predicted probability distribution $\hat{\boldsymbol{y}}^{(h)}$, and the true next word distribution $\boldsymbol{y}^{(h)}$, which is simply the one-hot vector for x_{m_h}. Therefore, we have

$$\begin{aligned} \mathcal{L}_{\text{MLM}}^{(h)}(\theta) &= \text{CE}(\boldsymbol{y}^{(h)}, \hat{\boldsymbol{y}}^{(h)}) \\ &= \sum_{w \in V} -y_w^{(h)} \log \hat{y}_w^{(h)} \\ &= -\log \hat{y}_{x_{m_h}}^{(h)} . \end{aligned} \tag{3.13}$$

The overall loss of the sequence is simply the average loss for all the k masked-out tokens in $\boldsymbol{x}^{masked}$,

$$\begin{aligned} \mathcal{L}_{\text{MLM}}(\theta) &= \frac{1}{k} \sum_{h=1}^{k} \mathcal{L}_{\text{MLM}}^{(h)}(\theta) \\ &= \frac{1}{k} \sum_{h=1}^{k} -\log \hat{y}_{x_{m_h}}^{(h)} \\ &= \frac{1}{k} \sum_{i \in \boldsymbol{m}} -\log p\left(x_i | \boldsymbol{x}^{\text{masked}}\right) . \end{aligned} \tag{3.14}$$

The masking procedure works as follows: BERT selects 15% of all WordPiece tokens in each training sequence at random. If the i-th token is chosen, it is replaced with:

1. the `[MASK]` token 80% of the time;
2. a random token 10% of the time;
3. the unchanged i-th token 10% of the time.

The selected words are not always replaced with the `[MASK]` token because it would then create a mismatch between pre-training and fine-tuning, since the masked token would never be seen before fine-tuning.

3.3.2 Next Sentence Prediction

Next sentence prediction (NSP) is a binary classification task in which the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original corpus. This training objective helps understand the relationship between pairs of sentences, which is not directly captured by language modeling but still very important for many downstream tasks such as question-answering (QA) and natural language inference (NLI).

This prediction task can can be easily generated from any monolingual corpus. Specifically, when choosing the sentences `A` and `B` for each pre-training example, 50% of the time `B` is the actual next sentence that follows `A` (labeled as `IsNext`), and the other 50% of the time it is a random sentence from the corpus (labeled as `NotNext`). In this case, the final hidden vector corresponding to the `[CLS]` token is fed into an output softmax over the two possible predictions, as shown in Figure 3.12.

Figure 3.12: Illustration of the *next sentence prediction* (NSP) training objective (Jay Alammar, 2019).

3.4 Downstream Tasks

There are two strategies for applying pre-trained language representations to downstream NLP tasks: *fine-tuning* and *feature-based*. On one hand, the fine-tuning approach introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning all pre-trained parameters. On the other hand, the feature-based approach uses task-specific architectures that include the pre-trained representations as input features for learning the task.

3.4.1 Fine-tuning Approach

The two pre-training objectives of BERT allow it to be used on any single sequence and sequence-pair tasks without substantial task-specific architecture modifications. For each task, one only

needs to plug in the task-specific inputs and outputs into BERT and fine-tune all the parameters end-to-end for a few epochs. Figure 3.13 illustrates the fine-tuning of BERT on different common tasks.

Figure 3.13: Illustrations of fine-tuning BERT on different tasks (Devlin et al., 2018).

At the input, sentence `A` and sentence `B` from pre-training are similar to:

1. sentence pairs in *paraphrasing*;
2. hypothesis-premise pairs in *entailment*;
3. question-passage pairs in *question-answering*;
4. a degenerate text-∅ pair in *text classification* or *sequence tagging*.

At the output, the token representations are fed into an output layer for token-level tasks (e.g., sequence tagging or question-answering), and the `[CLS]` representation is fed into an output layer for text classification (e.g., entailment or sentiment analysis).

3.4.2 Feature-based Approach

In addition to the fine-tuning approach, where a simple output layer is added to the pre-trained model and all parameters are jointly fine-tuned on a downstream task, BERT can also be used with a feature-based approach, where word representations are extracted from the pre-trained model and serve as inputs to other task-specific architectures. This approach has certain advantages over the fine-tuning one. First, not all tasks can be easily represented by a Transformer

encoder architecture, and therefore require a task-specific model architecture to be added. Second, there are major computational benefits to pre-compute an expensive representation of the training data once and then run many experiments with cheaper models on top of this representation.

There are several ways of extracting contextual word embeddings from BERT representations, and which approach works best mainly depends on the task it is being evaluated on. For example, Devlin et al. (2018) led a study for the task of named entity recognition (NER), where they applied a feature-based approach by extracting the activations from one or more layers without fine-tuning any parameters of BERT on the task, and then used these contextual embeddings as inputs to a randomly initialized two-layer 768-dimensional BiLSTM before the classification layer. The results showed that concatenating the last four hidden layers as the contextual word embeddings led to the best F1 scores for that specific task, as shown in Figure 3.14.

Figure 3.14: Results on named entity recognition (NER) using BERT embeddings with a feature-based approach (Jay Alammar, 2019).

Chapter 4

NetBERT

This chapter covers the pre-training of a novel language representation model for computer networking, called NetBERT. It is divided into three sections. First, Section 4.1 motivates the choice of using BERT over other existing NLP techniques. Then, Section 4.2 introduces the collection and processing of the large computer networking text corpus used for pre-training our novel model. Finally, Section 4.3 describes some strategies for efficient pre-training, as well as the training setup and results.

4.1 Model Choice

BERT is far from being the only model giving impressive results on language tasks. Lately, the field of NLP has been the subject of growing interest since the emergence of the Transformer architecture (Vaswani et al., 2017), which completely revolutionized the way of doing machine translation. While BERT was one of the first model to use a pre-trained Transformer encoder as a building block to perform state-of-the-art on several NLP tasks outside of machine translation, new models have since appeared almost every month, tweaking alternately some aspects of the initial network to achieve greater and greater results in many language tasks. Such models include GPT (Radford et al., 2018), Transformer-XL (Dai et al., 2019), XLM (Lample and Conneau, 2019), GPT-2 (Radford et al., 2019), ERNIE (Zhang et al., 2019a), XLNet (Yang et al., 2019), RoBERTa (Liu et al., 2019a), ERNIE 2.0 (Sun et al., 2019) and CTRL (Keskar et al., 2019). Nevertheless, BERT was chosen for this thesis because of three main reasons that are explained in this section.

4.1.1 The Google Search Update

Going back to the original problem of poor search results, presented in Section 1.1, one might wonder how the Cisco search engine could be improved. The best way to look at this question is to understand what makes a given search engine better than another. Currently, Google Search dominates the search engine market with an estimated market share of 92%, as shown in Figure 4.1. In late October 2019, Google made what they called the biggest change to Google search in the past 5 years, the so-called "BERT update". According to the company, this update affects both ranking and featured snippets in Search, and helps better understand one out of 10 searches in the U.S. in English[1]. Examples that demonstrate BERT's ability to understand the intent behind a query are given in Appendix A.3. The fact that the best search engine on the market makes such a change on its gem is an incentive, according to our opinion, to take a greater interest to that particular model for our related purpose.

[1] https://www.blog.google/products/search/search-language-understanding-bert/

Figure 4.1: Search Engine Market Share in 2020.[2]

4.1.2 BERT as Knowledge Base

Lately, research has been made on the viability of using state-of-the-art pre-trained language models (especially BERT) as factual knowledge bases. For example, Liu et al. (2019b) built a knowledge base question-answering (KBQA) system by using a pre-trained BERT model, which leverages prior linguistic knowledge to obtain deep contextual representations. Their experimental results showed state-of-the-art performance on KBQA datasets. In the same vein, Petroni et al. (2019) found that a pre-trained BERT model contains relational knowledge competitive with traditional NLP methods that have some oracle knowledge, demonstrating a huge potential as an unsupervised question-answering (QA) system. Poerner et al. (2019) questioned this evidence by showing that the surprising performance of BERT on QA benchmark is partly due to reasoning about entity names rather than factual knowledge (e.g., guessing that a person with an Italian-sounding name speaks Italian). As a remedy, they proposed a simple extension of BERT that replaces entity mentions with symbolic entity embeddings. They showed that their extension, E-BERT, outperforms BERT on hard-to-guess queries. Going even further, Peters et al. (2019) proposed a general method to embed multiple knowledge bases (including WordNet[3] and a subset of Wikipedia) into BERT, and thereby enhance their representations with structured human-curated knowledge. Their knowledge enhanced model, KnowBERT, demonstrates improved ability over BERT to recall facts.

These encouraging results obviously strengthened our choice of using BERT for this thesis, as our novel BERT-based model could then potentially be used as a factual knowledge base to solve the search problem discussed in Section 1.1.

4.1.3 Model Size

In general, as with many other deep learning models, the bigger the model, the better the performance. Hence, the NLP research community has recently begun a frantic race for top

[2] https://gs.statcounter.com/search-engine-market-share

[3] WordNet is a lexical database of semantic relations between words in more than 200 languages. For further information, see: https://wordnet.princeton.edu/.

performance by implementing larger and larger language models. For example, during the summer of 2019, researchers at NVIDIA announced Megatron-LM (Shoeybi et al., 2019), a massive Transformer-based model with 8.3 billion parameters (i.e., 75 times larger than BERT-base) that achieved state-of-the-art performance on a variety of language tasks. While this was undoubtedly an impressive technical achievement, the parameters alone weigh in at just over 33 GB on disk and pre-training the final model took 512 V100 GPUs running continuously for 9.2 days. One month later, Google AI researchers introduced T5 (Raffel et al., 2019), a text-to-text Transformer model with 11 billion parameters. Earlier this year, Microsoft Research announced Turing-NLG (Rosset, 2019), a 17-billion-parameters Transformer-based generative language model. Lastly, OpenAI researchers announced in late May GPT-3 (Brown et al., 2020), the largest language model ever seen at the time of writing with a total of 175 billion parameters, i.e., nearly 1,600 times larger than BERT-base. Figure 4.2 shows the size of some of the most popular NLP models in the past two years.

Figure 4.2: Parameter counts of popular pre-trained language models released in the past two years.

At some point, researchers have begun to realize that this trend towards bigger models might raise several concerns (Lan et al., 2019; Sanh et al., 2019; Turc et al., 2019). First, it hinders democratization. Indeed, if we believe in a world where thousands of engineers are going to use deep learning to make every application and device better, we won't get there with massive models that take large amounts of time and money to train. Second, it restricts scale. As a matter of fact, there are about 100 million servers that are currently being used all around the world,[4] but there are already 3.5 billion mobile phones[5] and 22 billion IoT devices.[6] In the long term, it will be these small, low power devices that will use the most deep learning, and massive models will simply not fit into them.

For this thesis, the computational power at our disposal was limited to one machine of

[4] https://www.racksolutions.com/news/data-center-trends/400-million-new-servers-might-be-needed-by-2020/

[5] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[6] https://www.statista.com/statistics/802690/worldwide-connected-devices-by-access-technology/

8×32GB NVIDIA Tesla V100 GPUs. While this may seem a lot in general, it is not for pre-training a large Transformer-based language model. Basically, it allows to efficiently pre-train "small" ($\sim$ 100M parameters) Transformer-based networks on a corpus of about 15GB with good parallelization algorithms within 11 days.[7] While pre-training "medium" ($\sim$ 350M parameters) models is technically possible, it would lead to a much longer training time (about 21 days with the same configuration) due to the reduced batch size that can fit in memory. Finally, pre-training "large" ($>$ 500M parameters) models is simply not feasible with the computational power and time available. By excluding all "medium" and "large" pre-trained models, as well as those that were exclusively pre-trained on a specific task such as machine translation, the remaining candidates included BERT-base (110M), XLNet-base (110M) and ERNIE 2.0 (114M).

Eventually, the extensive documentation on BERT, its open-source code and the proofs (Liu et al., 2019a) that an optimized version of its pre-training procedure made it competitive with more recent state-of-the-art methods (such as XLNet) motivated the choice of using BERT-base for this thesis, as for other related works (see Section 2.3).

4.2 Pre-training Data

4.2.1 Data Collection

The domain-specific corpus was collected by scraping all the text content from `wwwin.cisco.com`, the Cisco confidential employee website. It resulted in about 30GB of uncleaned text, collected from 442,028 web pages in total, and gathered into thirteen large JSON files where each item corresponds to one text document (i.e., the text extracted from one web page) and appears in the form `{'uri':"...", 'text':"..."}`. Note that the corpus had already been collected previous to this work. However, no cleaning or pre-processing had been performed on it before.

4.2.2 Data Preparation

The data preparation of the original uncleaned 30GB text corpus consists in two main cleaning stages. The first one is a high-level cleaning step focused on the collected documents, while the second one is a low-level cleaning step focused on individual sentences.

High-level Cleaning

This first cleaning step is in charge of making a first selection among the original 442,028 web pages, also referred as documents. It is inspired from the text cleaning process used for Megatron-LM (Shoeybi et al., 2019). It performs the following operations:

- Correct malformed documents, using the `ftfy` Python library.[8] This library allows to fix bad Unicode in text (e.g., the word *"schön"* might appear as *"schÃ¶n"* due to an incompatibility between the encoding-decoding standards).
- Remove non-English documents, using the `langdetect` Python library[9] (adapted from Google's `language-detection` library in Java). This operation allows the model not to be confused with a minority of documents written in a foreign language, while the vast majority appears in English, as shown in Table 4.1.
- Remove short documents of less than 128 tokens, as it appeared that such documents are mainly contact information, copyrights, references or a variety of subsequent text symbols and HTML tags. As these do not bring valuable information for pre-training the model, they are removed.

[7] https://timdettmers.com/2018/10/17/tpus-vs-gpus-for-transformers-bert/
[8] https://ftfy.readthedocs.io/en/latest/
[9] https://pypi.org/project/langdetect/

Table 4.1 summarizes the number of documents impacted by each cleaning operation described previously. Using multi-processing with an Intel Xeon Processor E5-2698 v4 (50M Cache, 2.20 GHz, 20 cores), it took about 0.08s to process one document (i.e., 9.5 hours for processing all documents).

Original documents	442,028
Small documents	14,853
Non-English documents	593
Fixed documents	367,568
Saved documents	426,582

Table 4.1: Statistics about the high-level cleaning step on the original documents.

Low-level Cleaning

The second cleaning stage takes care of removing or correcting individual sentences. To do so, each document is first split into sentences. This operation is performed using the `nltk` library.[10] Then, the following operations are performed on each individual sentence:

- Remove sequences of more than three consecutive special symbols from the following list: {, ?; . :!/ = +% ∗ −_()[]”ˆ <> •°&#@$£€©}. Such sequences (e.g., “=====”) often appear in figures, tables or tables of contents.
- Remove numbers or special characters appearing at the beginning of the sentence (e.g., “•” or “1.1”). These appear at the beginning of sections and subsections, as well as in ordered and unordered lists.
- Remove sentences of less than 2 words and more than 200 words. Very short sentences were mainly letters or numbers from ordered and unordered lists surrounded by brackets (e.g., “((*a*))” or “[2.3]”). Very long sentences were often complex text structure such as pieces of code (e.g., “*<?xml version="1.0" encoding="UTF-8"?><note>...</note>*”), tables or figures which were not correctly handled by the sentence tokenization.

These elements are removed from the sentences as they don’t bring any valuable information for learning the domain-specific language (on the contrary, they tend to add noise in the corpus).

4.2.3 Processed Dataset

The pre-processing of the original corpus results in a cleaned dataset of about 170.7M sentences, for a total size of 22.7GB. This dataset is further split into train/validation/test sets with a ratio 90%-5%-5% respectively. Statistics about the processed dataset is given in Table 4.2.

Data set	Number of sentences	Number of words	Vocabulary size (unique words)	Data size
Train	153.5M	3.3B	4.7M	20.4GB
Validation	8.8M	0.2B	1.6M	1.2GB
Test	8.4M	0.2B	1.5M	1.1GB

Table 4.2: Statistics about the processed `cisco.com` text corpus.

[10] https://www.nltk.org/

Lastly, Table 4.3 lists the different corpora that were used directly or indirectly for achieving NetBERT (as discussed in Section 4.3.1, NetBERT starts its pre-training from BERT weights, and therefore benefits from BERT's pre-training on general-domain corpora).

Corpus	Number of words	Data size	Domain
English Wikipedia	2.5B	12.4GB	General
BooksCorpus[11]	0.8B	3.6GB	General
`cisco.com`	3.7B	22.7GB	Computer networking

Table 4.3: List of text corpora used for achieving NetBERT. The English Wikipedia & BooksCorpus datasets were used to pre-train the original BERT, from which NetBERT extends the pre-training with the `cisco.com` corpus.

4.3 Pre-training

4.3.1 Strategies

Given the large expected pre-training time of the model (estimated to about one month given the size of the data and the computational power at our disposal), it was clear from the beginning that efficient pre-training strategies were necessary to achieve the best model possible in one unique pre-training run. These strategies were chosen according to an extensive review of previous related work.

Next Sentence Prediction

In the original BERT pre-training procedure, the model observes two concatenated sentences, which are either subsequently sampled from the same document (with $p = 0.5$) or randomly sampled from the whole corpus. In addition to the masked language modeling (MLM) objective (see Section 3.3.1), the model is trained to predict whether the observed sentences entail each other via an auxiliary next sentence prediction (NSP) loss (see Section 3.3.2). The NSP loss was hypothesized to be an important factor in pre-training the original BERT model, as Devlin et al. (2018) observed that removing NSP hurts performance in multiple downstream tasks. However, some recent work has questioned the necessity of this training objective (Lample and Conneau, 2019; Yang et al., 2019; Joshi et al., 2020). By comparing different BERT-based models pre-trained with and without the NSP loss, Liu et al. (2019a) eventually proved that removing it matches (or slightly improves) performance on downstream NLP tasks, while reducing computational cost. *Given these conclusions, NetBERT was trained on masked language modeling only.*

Static vs. Dynamic Masking

In the original BERT implementation, random masking (see Section 3.3.1) is performed once on the whole corpus during data pre-processing, resulting in a single *static* masked dataset. To avoid using the same mask for each training instance in every epoch, pre-training data was duplicated 10 times so that each sequence is masked in 10 different ways (resulting in a pre-training dataset of about 150GB from to the original corpus of 15GB). In contrast, RoBERTa (Liu et al., 2019a), a BERT-based model pre-trained with a robust approach, experimented a *dynamic* masking strategy, where the masking pattern was generated every time a sequence was fed into the model. They showed that dynamic masking was comparable or slightly better than

[11] BooksCorpus is a dataset consisting of 11,038 unpublished books from 16 different genres.

static masking when compared on downstream tasks performance, while significantly reducing the amount of training data through the removal of the data duplication process. *Given these results, NetBERT was trained using dynamic masking.*

Scratch vs. Checkpoint Pre-training

Intuitively, it seems more advantageous to start the domain-specific pre-training of our novel model from BERT pre-trained weights rather than starting the pre-training from scratch. Indeed, with the former option, the already learned word/token embeddings are only fine-tuned to adapt to the domain, while these embeddings need to be learned from the very beginning with the latter approach. Hence, we utilize knowledge from the general-domain model to improve learning in the target domain-specific model, benefiting from transfer learning. Regarding related work, both ClinicalBERT (Huang et al., 2019) and BioBERT (Lee et al., 2020) were initialized with BERT pre-trained parameters, whereas SciBERT (Beltagy et al., 2019) experimented a pre-training from scratch and another one from BERT weights, and eventually noticed very few changes in performance when evaluated on downstream tasks. *For these reasons, NetBERT was initialized with the pre-trained BERT-base parameters.*

Custom vs. General Vocabulary

The original BERT model uses WordPiece tokenization (see Section 3.2.2) whose fixed-size vocabulary was created on general-domain corpora (namely, English Wikipedia). At first glance, it might seem useful to learn a new WordPiece vocabulary built specifically on the domain-specific corpus. Indeed, this corpus contains a number of in-domain proper nouns (e.g., IEEE, LAN, SSL) and terms (e.g., Ethernet, WiFi) that are frequently used in the field and therefore would probably appear as they are in a new domain-specific vocabulary. As a result, while most in-domain words are split into subwords with a general-domain vocabulary, they would instead be used directly with a domain-specific vocabulary, which might possibly lead to a better understanding of these terms when learning the word representations. However, doing so did not prove to be so useful in practice. Indeed, Beltagy et al. (2019) initially assessed the importance of a domain-specific WordPiece vocabulary for pre-training a BERT-based model on scientific corpus. To prove it, they trained two versions of their model: one with a domain-specific vocabulary created from a scientific text corpus, and another one with the general-domain vocabulary released with BERT. They eventually found that the optimal hyperparameters of both versions often coincided. They therefore concluded that, while an in-domain vocabulary might possibly be helpful, the benefits of their domain-specific model mostly came from the scientific corpus pre-training. Moreover, as Lee et al. (2020) pointed out to justify their choice of using the general-domain BERT vocabulary for pre-training their BioBERT model, doing so allows to reuse the already largely pre-trained BERT checkpoint. In contrast, new vocabulary necessarily means pre-training from scratch, as the token embedding weight matrix (see Section 3.2.2) then refers to different tokens. Therefore, using a custom vocabulary makes it impossible to benefit from the transfer learning properties of a pre-trained BERT checkpoint. *For these reasons, NetBERT was pre-trained using the general-domain WordPiece vocabulary released with BERT.*

Cased vs. Uncased Vocabulary

With BioBERT, Lee et al. (2020) found out that using a cased vocabulary resulted in slightly better performance on downstream tasks compared to a lower-cased vocabulary. *Taking their results into consideration, NetBERT uses the cased version of the general-domain WordPiece vocabulary released with BERT.*

4.3.2 Setup

Implementation

NetBERT was implemented using HuggingFace's Transformers library (Wolf et al., 2019). This library features carefully crafted implementations as well as high-performance pre-trained weights of state-of-the-art NLP models (e.g., BERT, GPT-2, RoBERTa, XLM, XLNet, CTRL, etc) for two main deep learning frameworks: PyTorch and TensorFlow. It also supports all the necessary tools to analyze, evaluate and use these models in downstream tasks.

Hyperparameters Optimization

The hyperparameters primarily followed those from the original BERT implementation. More specifically, the model was optimized with Adam (Kingma and Ba, 2014) using the following parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, ϵ =1e-8 and L_2 weight decay of 0.01. The learning rate was warmed up over the first 10,000 steps to a peak value of 5e-5, and then linearly decayed.[12] It trained with a dropout of 0.1 on all layers and attention weights, and a GELU activation function (Hendrycks and Gimpel, 2016). The mini-batch size was set to $B = 112$ sequences of maximum length $T = 512$ tokens.

Hardware and Schedule

The model pre-training was performed on one machine with 8 $\times$ 32GB NVIDIA Tesla V100 GPUs. Using the hyperparameters described previously, each training step took about 1.18 seconds. The model trained continuously for 20 epochs (i.e., 1.9M training steps) which took a total of 29 days.

4.3.3 Results

The MLM training loss of NetBERT is shown in Figure 4.3. One can notice that the loss was constantly decreasing during pre-training, and this trend seemed to continue even in the last iterations. Arguably, pre-training the model longer would probably have led to a loss that would have continued to decrease. For comparison, BERT was pre-trained for 40 epochs (i.e., on twice as many epochs as NetBERT). However, the time available didn't allow for a longer pre-training. Moreover, this training was sufficient to motivate improvements over the base model, as discussed in the rest of this thesis.

Figure 4.3: Masked language modeling (MLM) training loss of NetBERT. In the figure, 200k iterations equals 3 days of continuous training on 8 $\times$ 32GB NVIDIA Tesla V100 GPUs.

[12] Although the original learning rate used for pre-training BERT was 1e-4, Devlin et al. (2018) recommended to use a smaller one (e.g., 2e-5) when performing additional pre-training steps from an existing BERT checkpoint.

Once pre-trained, NetBERT was evaluated and compared to BERT. The standard evaluation metric for language models (LM) – i.e., models that predict the next word of a sequence given the previous words from that sequence – is perplexity. Formally, the perplexity (ppl) of a language model on a corpus W (composed of T words from vocabulary V) corresponds to the inverse probability of the corpus normalized by the number of words in the corpus,

$$\begin{aligned}\text{ppl}(W) &= p_{LM}\left(x^{(1)}x^{(2)}\dots x^{(T)}\right)^{-\frac{1}{T}} \\ &= \prod_{t=1}^{T} p_{LM}\left(x^{(t+1)}|x^{(t)}\dots x^{(1)}\right)^{-\frac{1}{T}},\end{aligned} \tag{4.1}$$

where the equality is obtained using the chain rule. Furthermore, we can show that the perplexity of a model on corpus W is equal to the exponential of the LM loss,

$$\text{ppl}(W) = \exp\left(\mathcal{L}_{LM}(\theta)\right). \tag{4.2}$$

As a reminder, the LM loss function on step t is the cross-entropy between predicted probability distribution $\hat{\boldsymbol{y}}^{(t)}$ and the true next word distribution $\boldsymbol{y}^{(t)}$, which is simply a one-hot vector for token $x^{(t+1)}$. Hence, we get

$$\begin{aligned}\mathcal{L}_{LM}^{(t)}(\theta) &= \text{CE}(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) \\ &= -\sum_{w\in V} y_w^{(t)} \log \hat{y}_w^{(t)} \\ &= -\log \hat{y}_{x_{t+1}}^{(t)}.\end{aligned} \tag{4.3}$$

The overall loss is then the average of the latter expression over the entire corpus,

$$\begin{aligned}\mathcal{L}_{LM}(\theta) &= \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}_{LM}^{(t)}(\theta) \\ &= \frac{1}{T}\sum_{t=1}^{T} -\log \hat{y}_{x_{t+1}}^{(t)}.\end{aligned} \tag{4.4}$$

Finally, the expression of the perplexity in Equation (4.1) can be rewritten as

$$\begin{aligned}\text{ppl}(W) &= \prod_{t=1}^{T}\left(\hat{y}_{x_{t+1}}^{(t)}\right)^{-\frac{1}{T}} \\ &= \exp\left(\log\left(\prod_{t=1}^{T}\left(\hat{y}_{x_{t+1}}^{(t)}\right)^{-\frac{1}{T}}\right)\right) \\ &= \exp\left(\frac{1}{T}\sum_{t=1}^{T} -\log \hat{y}_{x_{t+1}}^{(t)}\right) \\ &= \exp\left(\mathcal{L}_{LM}(\theta)\right).\end{aligned} \tag{4.5}$$

Similarly, the perplexity of a masked language model (MLM) – i.e., a model that predicts a percentage of input tokens which were randomly masked – over a masked corpus M (where k tokens were randomly masked at positions m_h, $h = 1, ..., k$) is defined as the exponential of the MLM loss (see Section 3.3.1),

$$\begin{aligned}\text{ppl}(M) &= \exp\left(\mathcal{L}_{MLM}(\theta)\right) \\ &= \exp\left(\frac{1}{k}\sum_{h=1}^{k} -\log \hat{y}_{x_{m_h}}^{(h)}\right).\end{aligned} \tag{4.6}$$

In order to perform a first comparison between BERT and NetBERT, perplexity scores were computed for both models on the train, validation and test sets. Results are shown in Table 4.4. It appears that NetBERT largely outperforms BERT on the task of masked language modeling when evaluated on domain-specific (i.e., computer networking) text corpus.

		NetBERT		
Data set	BERT	3 epochs	12 epochs	20 epochs
Train	34.618	1.423	1.298	**1.253**
Validation	34.674	1.420	1.302	**1.258**
Test	34.456	1.416	1.302	**1.259**

Table 4.4: Perplexity scores of NetBERT and BERT on train/validation/test sets. **Bold** indicates the best results.

Chapter 5

Experiments

Pre-trained language models return contextual word representations. As part of this thesis, we want to study if the representations of our novel model (NetBERT) pre-trained on domain-specific corpora better capture both syntactic and semantic meanings of that domain, compared to those of a general-domain model (BERT). There exist various evaluation methods for testing the quality of word embedding models (Bakarov, 2018). However, all these methods basically fall into two main categories: *extrinsic* and *intrinsic* evaluations. On one hand, extrinsic evaluators use word embeddings as the features vectors for downstream NLP tasks. In that case, performance is being measured on a dataset for the given NLP task and is often perceived as a measure of word embedding quality. In practice, researchers assume that word embeddings showing a good result on one task will show a good result on others (i.e., the results of word embeddings on different tasks correlate). On the other hand, intrinsic evaluators are experiments in which word embeddings are compared to human judgments on words relations. These evaluators allow to measure the ability of an embedding model to capture meaningful semantic properties within their representations.

This chapter presents several extrinsic and intrinsic evaluations in order to compare BERT to NetBERT on domain-specific language tasks. In Section 5.1, both models are fine-tuned on the task of domain-specific *text classification* and then deeply compared on their respective performance. Then, Section 5.2 focuses on domain-specific *information retrieval*, where both models are evaluated in their ability to retrieve text chunks corresponding to a given query. Next, Section 5.3 describes a *word similarity* evaluation to identify the capacity of both models to capture the different meanings of a same domain-related word. Finally, Section 5.4 details a *word analogy* evaluation that investigates the ability of both models to capture relevant relationships between networking concepts.

5.1 Text Classification

This task aims at comparing the quality of both BERT and NetBERT embeddings by fine-tuning both models on a domain-specific *sentence classification* task. Intuitively, if one model is able to predict the class of a sentence better than the other, it means that the word representations that have been learned by the former model capture a more accurate meaning of that sentence than those learned by the other model.

In the following, Section 5.1.1 describes the dataset used to conduct this experiment. Then, Section 5.1.2 explains the approach used for fine-tuning both models on the task of domain-specific sentence classification. Next, Section 5.1.3 discusses and compare the overall performance of both models. Furthermore, Section 5.1.4 conducts an extensive analysis on the performance of both models, and investigates the potential improvements of NetBERT over BERT. Finally, Section 5.1.5 concludes this experiment by discussing the key elements to be drawn from it.

5.1.1 Dataset

The dataset used in this experiment was collected by the Cisco One Search team in San Jose, California. They gathered a set of actual search queries from Cisco employees, and labeled them with the type of document in which the information being sought was found. In total, the dataset contains about 48,000 queries labeled with seven different document types, described in Table 5.1.

To put it into context, the Cisco One Search team is currently working on the improvement of the Cisco search engine. Recently, they experimented with integrating BERT into their system as a "query → document type" classifier, allowing to retrieve a first good set of document candidates that best match a given query. The results were very encouraging. Hence, we decided to learn a similar classifier with NetBERT and compare the performance of both models.

Document type	**Description**
Configuration	These documents characterize the information that defines the performance, functional and physical attributes of a product.
Install & Upgrade Guides	These documents explain pre-installation, installation, post-installation and upgrading procedures of a product or service.
Data Sheets	These documents summarize the performance and other characteristics of an item or product, and is usually used for commercial or technical communication.
Release Notes	These documents detail the corrections, changes or enhancements made to the service or product the company provides.
End User Guides	These documents are designed to assist end users to use the product or service.
Maintain & Operate	These documents describe all measures required to ensure and maintain the functional capability of a system or product.
Command References	These documents provide pertinent details for consultation about a subject.

Table 5.1: Description of the different document types in the query classification task.

Figure 5.1 shows the number of queries per document type. Here, we hypothesize that classes *Maintain & Operate* and *Command References* do not have enough samples to effectively train a classifier on them. Hence, only the top five classes in terms of number of queries are considered in this experiment (i.e., *Configuration*, *Install & Upgrade Guides*, *Data Sheets*, *Release Notes* and *End User Guides*). Also, one can notice that the classes are highly imbalanced, which will be taken into account when evaluating performance (by considering adequate classification metrics).

Figure 5.1: Number of samples in each class of the query classification dataset.

Finally, the resulting dataset was randomly split into train (80%), validation (10%) and test (10%) sets. The distribution among classes of each set is given in Table 5.2.

	Train	Val	Test
#Configuration	24,121	3,010	3,022
#Install & Upgrade Guides	5,655	705	737
#Data Sheets	4,124	502	513
#Release Notes	3,036	411	345
#End User Guides	1,522	180	191

Table 5.2: Samples distribution among classes in the train, validation and test sets.

5.1.2 Fine-tuning

Fine-tuning a BERT-based model is pretty straightforward and relatively inexpensive compared to the pre-training step. Basically, one simply needs to plug in the task-specific inputs and outputs into a pre-trained BERT model, and fine-tune all the parameters end-to-end for a few epochs. For the given task of sentence classification, the inputs stay unchanged. At the output, the `[CLS]` representation is fed into an output layer for classification, which is a simple feed-forward neural network followed by a softmax operation. The final output represents the predicted probability distribution of the document types, as illustrated in Figure 5.2.

Figure 5.2: Illustration of fine-tuning NetBERT on text classification.

For fine-tuning, most model hyperparameters are the same as in pre-training, with the exception of the batch size, learning rate, and number of training epochs. The optimal hyperparameter values are task-specific. However, Devlin et al. (2018) found the following range of possible values to work well across most tasks: *batch size* = {16, 32}, *learning rate* = {5e-5, 3e-5, 2e-5} (Adam), *number of epochs* = {2, 3, 4}. In order to find the optimal hyperparameters for our task, an exhaustive search over these parameters (the number of epochs was extended to 6) was run for both BERT and NetBERT, and the models that perform best on the validation set were chosen.

The following metrics were considered for evaluation: *precision*, *recall* and *F1-score*. These metrics (whose computations are reminded in Appendix B.1.1) are commonly used for binary

classification problems. In order to compute a global score for each of these metrics in our multi-class classification task, two approaches were implemented: *macro-averaging* and *weighted-averaging*. Macro-averaging is simply an arithmetic mean of the per-class metrics,

$$x_M = \frac{1}{N} \sum_{i=1}^{N} x_i, \tag{5.1}$$

where N is the number of classes in the classification problem. With that approach, each class is given an equal weight. In contrast, weighted-averaging takes class imbalance into account by weighting the different metrics x_i of each class by the number of samples w_i from that class,

$$x_W = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i}. \tag{5.2}$$

The results of the search for optimal hyperparameters are given in Appendix B.1.2. When looking at macro-average scores, the best models were obtained by fine-tuning for 6 epochs with a batch size of 32 and a learning rate of 5e-5. Considering weighted-average scores, fine-tuning for 6 epochs with a batch size of 16 and a learning rate of 3e-5 led to the best models. Globally, both configurations achieved very similar performance. Eventually, it was decided to consider the optimal parameters regarding weighted-averaging, in order to take class imbalance into account.

5.1.3 Results

Both BERT and NetBERT were fine-tuned a last time on the train and validation sets with the optimal hyperparameters found previously (i.e., *batch size* = 16, *learning rate* = 3e-5, *number of epochs* = 6), and evaluated on the test set (that remained unused and hidden until the final evaluation). In addition to the *precision*, *recall* and *F1* scores computed either by a weighted or a macro average of the per-class metrics, the *accuracy* as well as the *Matthews correlation coefficient* (MCC) were also reported. Note that MCC is generally regarded as a balanced measure in classification problems, which can be used even if the classes are of very different sizes. It is said to be more informative than the F1-score for such problems (Chicco and Jurman, 2020), because it takes into account the balance ratios of the four confusion matrix categories (TP, TN, FP and FN). The detailed computation of the MCC is given in Appendix B.1.1.

In order to report some notion of variability on the computed metrics, both models were evaluated on 100 bootstrapped samples drawn from the test set (of the same size as the latter).[1] The mean values as well as the standard deviations are summarized (in percent) in Table 5.3.

		Model	
Average	Metrics	BERT	NetBERT
	MCC (SD)	88.3 (0.6)	**89.6** (0.6)
	Accuracy (SD)	93.4 (0.3)	**94.1** (0.3)
Weighted	Precision (SD)	93.4 (0.3)	**94.2** (0.3)
	Recall (SD)	93.4 (0.3)	**94.1** (0.3)
	F1 (SD)	93.4 (0.3)	**94.1** (0.3)
Macro	Precision (SD)	91.7 (0.5)	**92.1** (0.5)
	Recall (SD)	90.2 (0.7)	**91.6** (0.6)
	F1 (SD)	90.9 (0.5)	**91.8** (0.5)

Table 5.3: Comparison between BERT and NetBERT results (in percent) on the test set. Values are the mean results of 100 bootstrapped samples drawn from the test set; standard deviations (SD) are in parentheses. The best scores appear in **bold**.

[1] Bootstrapping is a resampling method that uses random sampling with replacement (Efron, 1992).

Clearly, NetBERT outperforms BERT for this task of domain-specific sentence classification. However, the improvement in performance is not that significant (+1.3% in MCC and +0.9% in macro-average F1). At first glance, it seems that although not having been pre-trained on computer networking corpora, BERT still does a very good job at classifying these domain-specific queries. In practice, this small improvement might not justify on its own the additional expensive pre-training of NetBERT.

5.1.4 Further Analysis

Previous results showed that NetBERT outperforms BERT when considering several classification metrics, each averaged over all classes. In this section, the predictions of both models are thoroughly analyzed by looking at each class individually. Three questions in particular are investigated:

1. How does NetBERT improve/decline over BERT in its classification ability when looking at classes individually?
2. Does NetBERT perform at least as well as BERT? In other words, does NetBERT classify correctly the queries that were properly classified by BERT?
3. How does NetBERT improve the predictions of BERT? In other words, in which proportion does NetBERT correctly classify the queries that were misclassified by BERT?

Question 1: How does NetBERT improve/decline over BERT in its classification ability when looking at classes individually?

The main idea of this investigation is to study if NetBERT performs better than BERT on some classes than others and if so, to try to figure out the reasons. A preliminary analysis of the confusion matrices of both models on the test set, presented in Figure 5.3, reveals that the per-class accuracy is relatively high regardless of the model, ranging between 88% and 92% (except for the *Configuration* class for which models reach an accuracy around 96%). This means that, for both models, no class seems to be specifically more difficult to predict than another.

(a) BERT confusion matrix.

(b) NetBERT confusion matrix.

Figure 5.3: Normalized confusion matrices of BERT and NetBERT on the test set.

A closer look at the matrices allows to observe that NetBERT improves the per-class accuracy by a relatively constant percentage (between +0.2% and +1.8%) for three classes (namely, *Install & Upgrade Guides*, *Configuration* and *Data Sheets*), while the *End User Guides* class is significantly improved by 4.2%. On the other hand, it turns out that BERT achieves a higher accuracy than NetBERT on the *Release Notes* class. This is shown more clearly in Figure 5.4.

Figure 5.4: Per-class accuracy (in percent) of BERT and NetBERT on the test set.

In order to get an insight of why NetBERT classifies the queries from the *End User Guides* class much better than BERT, some of these queries were sampled and analyzed. In particular, Table 5.4 shows a sample of queries that had been properly classified by NetBERT but misclassified by BERT. At first glance, one can notice that these queries are strongly related to computer networking with a very specific vocabulary, which might possibly explain the better performance of NetBERT on these given queries. However, it is important to note that a lot of domain-specific terms also appear in the queries of other classes. Hence, no well-defined pattern was found (by us and a few computer networking experts) in those queries to explain why NetBERT performs much better on this particular class rather than on the others.

Sample of *End User Guides* queries
- Collect NAM Data On Cisco Prime Infrastructure 2.0
- IPv4 Switching: Provider Edge Router over MPLS
- Cisco Hosted Unified Communications Services: Managing Legacy PBX Support
- dnac sensor https test
- Cisco Modeling Labs: Export the Configuration to SVG Files
- UCCX supervisor multiple team
- getting started guide extended enterprise

Table 5.4: Sample of test set queries from the *End User Guides* class, misclassified by BERT but correctly classified by NetBERT.

A similar investigation was made concerning the queries from the *Release Notes* class, for which BERT does a better classification job than NetBERT. Table 5.4 shows a sample of *Release Notes* queries that were misclassified by NetBERT, but correctly classified by BERT. The only noticeable difference with the sample of *End User Guides* queries seems to be the presence of well-formed questions about a specific product. However, similar questions also appear for queries of other classes and, here again, no specific pattern about the form or vocabulary of these queries was found to justify the decline in accuracy of NetBERT compared to BERT. A more detailed analysis carried out by networking experts is needed to try to determine such patterns.

Sample of *Release Notes* queries
- What is the License model for the Catalyst 9300 Series Switches ?
- Can i know Cat 9500 ?
- nexus 7010 show tech
- How do I troubleshoot the Catalyst 9500 ?
- Bios software nexus 5000
- nexus 7000 fabric power-dn

Table 5.5: Sample of test set queries from the *Release Notes* class, correctly classified by BERT but misclassified by NetBERT.

Another way to understand why one or the other model performs better on some classes than on others is to visualize the `[CLS]` token representation of each query in a lower dimensional space. As a reminder, this representation is used as unique input to the output layer for classification. Hence, by reducing the dimension of such vectors and further visualize them in 2D or 3D can allow the identification of some clusters within the cloud of data points, which should ideally represent the different classes. If two or more classes came to overlap, this could potentially explain why the classifiers have difficulties to properly predict these specific classes.

Dimensionality reduction was performed using t-Stochastic Neighbor Embedding (t-SNE; Maaten and Hinton, 2008), which is a probabilistic method that does non-linear dimensionality reduction, and is widely used to visualize high-dimensional data. In short, the t-SNE algorithm comprises two main stages. First, t-SNE constructs a probability distribution over pairs of high-dimensional objects in such a way that similar objects (according to the Euclidean distance) are assigned a higher probability while dissimilar points are assigned a very low probability. Second, t-SNE defines a similar probability distribution over the points in the low-dimensional map in such a way that it minimizes the Kullback–Leibler (KL) divergence between the two distributions. In practice, t-SNE uses quite computationally heavy methods and therefore has certain limitations. For example, if the number of features is very high, Maaten and Hinton recommend to use another dimensionality reduction method such as Principal Component Analysis (PCA; Pearson, 1901) to reduce the number of dimensions to a reasonable amount (e.g., 50) before applying t-SNE. As BERT and NetBERT vectors are represented in $\mathbb{R}^{768}$, this recommendation was taken into account. Hence, the `[CLS]` token representations of the test set queries from both fine-tuned models were reduced to 50-dimensional vectors with PCA. Then, these vectors were themselves reduced to 2 and 3 dimensions with t-SNE. The resulst are shown in Figure 5.5.

In general, it appears that the representations of both fine-tuned models lead to well-defined clusters representing each of the classes. A deeper look at Figure 5.5b reveals a clear overlap between the `[CLS]` token representations of the *Release Notes* and *Configuration* classes. In contrast, the representations of the *Release Notes* queries appear as a uniform cluster with BERT. This might possibly explain why BERT achieves better classification performance for that particular class. Also, it can be noticed that the representations of classes *Install & Upgrade Guides* and *Data Sheets* appear as more separate clusters when computed with NetBERT than with BERT, which might explain the $+\sim 1.5\%$ accuracy improvement of NetBERT on these classes. Concerning the *End User Guides* class, for which NetBERT greatly improves the accuracy by $+4.2\%$, no apparent explanation seems to be visible on these graphs. Indeed, the representations of these queries appear as a distinctive cluster for both fine-tuned models.

(a) 2D projection of the [CLS] token embeddings from fine-tuned BERT.

(b) 2D projection of the [CLS] token embeddings from fine-tuned NetBERT.

(c) 3D projection of the [CLS] token embeddings from fine-tuned BERT.

(d) 3D projection of the [CLS] token embeddings from fine-tuned NetBERT.

Figure 5.5: Visualisation of the [CLS] token representations of all test set queries embedded with both BERT and NetBERT, and reduced to a lower dimensional space (2D and 3D) with t-SNE.

Lastly, we compared the wrong predictions of BERT with those of NetBERT. More precisely, the comparison was on the types of query that were the most misclassified by both models. Figure 5.6 shows the distribution among classes of BERT and NetBERT wrong predictions on all test set queries. It appears that, in general, the query types that both models have the most difficulties to classify are the same and in relatively similar proportions. The only notable difference is that BERT misclassifies nearly as much the *End User Guides* queries as the *Release Notes* ones (7.6% and 9.8% of BERT's wrong predictions, respectively) while NetBERT does a much better job at classifying *End User Guides* queries compared to the *Release Notes* ones (5.6% against 11.9% of NetBERT's wrong predictions, respectively).

(a) BERT wrong predictions.

(b) NetBERT wrong predictions.

Figure 5.6: Comparison between BERT and NetBERT proportions of misclassified test set queries according to the type of document.

Question 2: Does NetBERT perform at least as well as BERT?

The second question of interest aims at studying if NetBERT manages to correctly classify the queries that were properly classified by BERT. To this end, all the queries from the test set that BERT correctly classified were gathered and fed into NetBERT for prediction. The results are shown in percent in Table 5.6. Additionally, the corresponding confusion matrix is shown in Figure 5.7.

		Weighted-average			Macro-average		
MCC	A	P	R	F1	P	R	F1
96.7	98.2	98.2	98.2	98.2	97.5	97.5	97.5

Table 5.6: Results of NetBERT on the test set queries that BERT correctly classified. Evaluation metrics are macro-average and weighted-average Precision (P), Recall (R) and F1-score (F1), as well as Matthews coefficient correlation (MCC) and overall Accuracy (A), all expressed in percent.

One can notice that NetBERT performs extremely well on the queries that BERT correctly classified, but not perfectly. Intuitively, it means that although NetBERT is better overall than BERT on this task of domain-specific query classification, BERT is still able to sometimes capture a better representation of some queries than NetBERT. This is especially true for the *Release Notes* queries for example, as mentioned previously. Indeed, we see that 4% of the *Release Notes* queries that were correctly classified by BERT are misclassified by NetBERT.

Figure 5.7: Normalized confusion matrix of NetBERT predictions on the test set queries *that BERT classified correctly.*

A closer inspection of the queries that NetBERT misclassified reveals that most of the time, NetBERT is wrong by predicting either the *Configuration* class or the *Install & Upgrade Guides* class, as shown in Figure 5.8. It turns out that these classes also represent the largest classes from the dataset, as presented in Figure 5.2. Hence, it may be that for queries where NetBERT is not able to predict the corresponding classes with certainty, it learned to simply predict one of the classes that appear the most in the dataset.

Figure 5.8: NetBERT predicted classes on the test set queries that BERT correctly classified but NetBERT misclassified.

Question 3: How does NetBERT improve the predictions of BERT?

The third and last question of interest aims at analyzing the improvements of NetBERT on the queries that BERT misclassified. For this purpose, all the queries from the test set that BERT misclassified were gathered and fed into NetBERT for prediction. The results are shown in percent in Table 5.7.

		Weighted-average			Macro-average		
MCC	A	P	R	F1	P	R	F1
12.6	35.3	35.9	35.3	35.5	39.1	36.1	37.3

Table 5.7: Results of NetBERT on the test set queries that BERT misclassified. Evaluation metrics are macro-average and weighted-average Precision (P), Recall (R) and F1-score (F1), as well as Matthews coefficient correlation (MCC) and overall Accuracy (A), all expressed in percent.

Additionally, the resulting confusion matrix is shown in Figure 5.9. There are several points to note here. First, the diagonal of the matrix reveals that NetBERT managed to correctly classify about 30% to 35% of the queries for most classes, which is pretty mediocre, but at least a little better than random (i.e., 20% in a 5-class classification problem). It can be noticed that the predictions on the *End User Guides* class achieved a much higher accuracy though (of about 46%), which confirms what has been concluded previously about NetBERT's ability to better predict this type of queries.

Another interesting observation concerns the upper-left corner of the confusion matrix, highlighting the predictions of both *Install & Upgrade Guides* and *Configuration* queries. One can see that NetBERT predicts almost as much the one as the other class for these two types of queries. But more than that, it is generally mistaken by predicting the *Configuration* class more for *Install & Upgrade Guides* queries and vice-versa. A similar behaviour can be observed for the *Data Sheets* queries, whose predictions are almost similarly spread into the *Install & Upgrade Guides*, *Configuration* and *Data Sheets* classes. This is illustrated more clearly in Figure 5.10.

Lastly, it can be seen that nearly half of the *Release Notes* queries are misclassified as *Configuration* queries, whereas only about one third of those queries are correctly classified. This highlights the fact that NetBERT does not perform better than BERT for classifying this particular type of query, as observed previously.

Figure 5.9: Normalized confusion matrix of NetBERT predictions on the test set queries that BERT misclassified.

Figure 5.10: Distribution among the classes predicted by NetBERT for the test set queries that BERT misclassified.

5.1.5 Conclusions

In this experiment, the performance of both BERT and NetBERT models were compared on the task of domain-specific text classification. To this end, the models were fine-tuned on a dataset of about 48,000 real search queries from Cisco employees labeled by the type of document in which the information being sought was found. Eventually, it was shown that NetBERT outperforms BERT for this task with an improvement of +1.3% MCC (Matthews correlation coefficient) and +0.9% (macro-average) F1.

A deeper investigation showed that NetBERT improves BERT predictions in a relatively similar way for most classes (between +0.2% and +1.8% improvement in per-class accuracy). Nevertheless, one of the classes saw a significant improvement of +4.6% in accuracy with Net-

BERT, while another one saw a decline of -0.9% compared to BERT's predictions. Further studies were made in order to understand why NetBERT performs better on the former type of queries and not as well on the latter one. Samples of queries related to these classes were analyzed on their form, syntax and vocabulary, but no obvious conclusions could be drawn from this analysis. However, by reducing the dimension of `[CLS]` token representations (used for classification) and plotting the lower dimensional vectors in 2D/3D allowed to get an intuition of these improvements and declines. Indeed, classes that saw an improvement with NetBERT are also represented by very distinctive clusters in the lower dimensional space, while some of these clusters are overlapping with BERT representations. Similarly, the class which saw a decline with NetBERT appears as a more uniform cluster with BERT than it does with NetBERT.

Then, a study was conducted to investigate if NetBERT managed to correctly classify the queries that had been properly classified by BERT. It was shown that NetBERT performed extremely well on these queries too, but not perfectly (96.7% and 97.5% in MCC and macro-averaged F1-score, respectively). Intuitively, it means that although NetBERT performs better overall than BERT at classifying domain-specific queries, the latter is still able to sometimes capture a better representation of some queries than NetBERT.

Finally, a last study was made to investigate how NetBERT improves the predictions of the queries misclassified by BERT. The results showed that NetBERT manages to correctly classify slightly more than one third of these queries, which is pretty mediocre but a little better than random. However, a deeper look at the per-class performance revealed that the model still misclassifies most classes overall.

In conclusion, this experiment showed that NetBERT outperforms BERT on the task of domain-specific text classification. However, the improvements are far from being significant. Actually, it turns out that although not having been pre-trained on computer networking corpora, BERT still does a very good job at classifying domain-specific queries. It is important to note that this conclusion is drawn from a single dataset experiment. Several similar experiments performed on other datasets would ideally be required to draw any general conclusion.

5.2 Information Retrieval

This experiment aims at comparing the quality of both BERT and NetBERT embeddings on the task of domain-specific *information retrieval*, using a simple similarity-based search approach (no further fine-tuning). Intuitively, if one model is able to retrieve better answers than the other given in-domain queries, it means that the word representations of that model better capture the meaning of both the queries and the candidate answers than those of the other model.

In the following, Section 5.2.1 first describes the dataset used in this experiment. Then, Section 5.2.2 explains the approach used for performing similarity-based information retrieval and attributing a score to answers. Next, Section 5.2.3 discusses and compares the performance of both models on the task. Furthermore, Section 5.2.4 conducts a brief analysis about some notable differences in the ability of both models to retrieve information. Finally, Section 5.2.5 ends this experiment by drawing a conclusion from the observed results.

5.2.1 Dataset

The dataset used in this experiment contains a set of domain-specific text chunks (one or multiple sentences) and questions collected from the Cisco CCNA Routing and Switching book.[2] This book is designed to provide a complete study system for the `Cisco ICND1 100-105` exam, which tests a candidate's knowledge and skills related to network fundamentals, LAN switching technologies, routing technologies, infrastructure services, and infrastructure maintenance. It is

[2] https://www.ciscopress.com/store/ccna-routing-and-switching-200-125-official-cert-guide-9781587205811

structured in 37 chapters for a total of nearly 890 pages (appendices excluded). The real value of this book for our purpose comes from the *"Do I Know This Already?"* quizzes, which appear at the beginning of each chapter. These quizzes contain between 4 to 10 multiple-choices questions intended to test the level of knowledge of the reader before he/she starts the chapter. Hence, all the questions were extracted together with their ground truth answer(s) – given at the bottom of the quiz pages – to form a dataset of about 200 in-domain questions-answers. These questions were further labeled by the type of questions they represent (namely, `Knowledge` or `Scenario`) and the type of their answers (`Unique` or `Multiple`). Table 5.8 gives a few examples of such questions with their corresponding type.

Question	Question type
Where does the Ethernet FCS field reside in?	`Knowledge`
Which OSI layer defines the standards for cabling and connectors?	`Knowledge`
What does the acronym VLSM stand for?	`Knowledge`
Fred has just added DSL service at his home, with a separate DSL modem and consumer-grade router with four Ethernet ports. Fred wants to use the same old phone he was using before the installation of DSL. What happens to the phone cabling and phone used with his new DSL installation?	`Scenario`
Imagine a switch with three configured VLANs. How many IP subnets are required, assuming that all hosts in all VLANs want to use TCP/IP?	`Scenario`

Table 5.8: Examples of questions from the *"Do I Know This Already?"* quizzes in the Cisco CCNA Routing and Switching book. These questions were labeled either by the `Knowledge` type when the answers are pure knowledge (no reflection involved) or by the `Scenario` type when the answers involve a reflection on domain knowledge.

In this experiment, we focused on the `Knowledge` questions, which are used as queries in this domain-specific information retrieval task. Indeed, most of the `Scenario` questions are not suitable for the task of information retrieval, as answers to this type of questions rarely appear as they are in the book as they require thinking in addition of knowing. Both BERT and NetBERT are designed to capture meaning, but are in no way capable of performing any kind of reflection. Therefore, it was deemed wise to leave these questions aside. Figure 5.11 shows the distribution of the collected questions by question and answer types.

(a) Number of questions per question and answer types.

(b) Percentage of questions per question type.

Figure 5.11: Statistics about the questions from the Cisco CCNA Routing and Switching book. Only the `Knowledge` questions are used in this experiment.

After creating the queries dataset, the Cisco CCNA book was entirely scraped, cleaned and split into text chunks of maximum length $T = 512$ tokens, which is the maximum number

of tokens that both BERT and NetBERT can process as inputs. Obviously, all the multiple-choices questions and their answers were removed beforehand. It resulted in about 1,020 text chunks representing the corpus in which the similarity-based information retrieval is performed. Additional statistics about that corpus is shown in Table 5.9.

Number of book pages	Number of sentences	Number of words	Number of text chunks
890	13.5K	308.5K	1.02K

Table 5.9: Statistics about the Cisco CCNA text corpus used for information retrieval.

5.2.2 Search Score

The main idea of the experiment is to perform a similarity-based search on each of the queries using on one hand BERT embeddings and on the other hand NetBERT embeddings, and then compute a score that reflects the quality of the search results in order to compare the performance of both models.

To that end, the text chunks gathered from the Cisco CCNA book are first encoded with both BERT and NetBERT. As a reminder, these models take a text chunk as input, and output a vector representation $\boldsymbol{x}_i \in \mathbb{R}^d$ for each token i of the input chunk (where $d = 768$ is the output representation dimension of the models). In order to get a representation of the full chunk, the most common technique is to average the last layer token representations, resulting in a vector $\overline{\boldsymbol{x}} \in \mathbb{R}^d$ such that

$$\overline{\boldsymbol{x}} = \frac{1}{T} \sum_{i=1}^{T} \boldsymbol{x}_i, \tag{5.3}$$

where T is the number of tokens in the input chunk. Similarly, all the queries are encoded with both models as d-dimensional vectors $\overline{\boldsymbol{q}} \in \mathbb{R}^d$, as the average of the token representations of the query,

$$\overline{\boldsymbol{q}} = \frac{1}{T} \sum_{i=1}^{T} \boldsymbol{q}_i. \tag{5.4}$$

Therefore, it results in a set of chunk representations $X_{\text{BERT}} = \{\overline{\boldsymbol{x}}_1, ..., \overline{\boldsymbol{x}}_N\}$ and query representations $Q_{\text{BERT}} = \{\overline{\boldsymbol{q}}_1, ..., \overline{\boldsymbol{q}}_m\}$ embedded with BERT, as well as similar sets $X_{\text{NetBERT}} = \{\overline{\boldsymbol{x}}'_1, ..., \overline{\boldsymbol{x}}'_N\}$ and $Q_{\text{NetBERT}} = \{\overline{\boldsymbol{q}}'_1, ..., \overline{\boldsymbol{q}}'_m\}$ embedded with NetBERT (where N is the total number of text chunks in the corpus and m the number of collected queries).

Given these sets, the top-k most similar chunks from X_{BERT} (respectively X_{NetBERT}) to each query in Q_{BERT} (respectively Q_{NetBERT}) can easily be retrieved by using a pre-selected distance function. This function is used to compute a distance between a query embedding and all chunk embeddings such that only the top-k chunks with the lowest distance are retrieved. More precisely, a query embedding $\overline{\boldsymbol{q}} \in Q_{\text{BERT}}$ (respectively $\overline{\boldsymbol{q}}' \in Q_{\text{NetBERT}}$) is first compared to all the chunk embeddings $\overline{\boldsymbol{x}}_j \in X_{\text{BERT}}$ (respectively $\overline{\boldsymbol{x}}'_j \in X_{\text{NetBERT}}$) for $j = 1, ..., N$ with respect to a given distance function $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$. Eventually, only the k-closest vectors to the query vector are returned, i.e., the sets

$$\begin{aligned} S_{\text{BERT}} &= k\text{-argmin}_j f(\overline{\boldsymbol{q}}, \overline{\boldsymbol{x}}_j) \\ S_{\text{NetBERT}} &= k\text{-argmin}_j f(\overline{\boldsymbol{q}}', \overline{\boldsymbol{x}}'_j). \end{aligned} \tag{5.5}$$

The distance function f is typically the *Euclidean* (L2) distance. Alternatively, a similarity function $s : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ can also be used instead of a distance function, in which case the k-argmax$_j$ is then computed to get the k-closest vectors. Commonly used similarity functions are the *dot-product* similarity and the *cosine* similarity.

The last step consists in evaluating the quality of the search results from both models. To do so, a retrieval score is computed for each top-k result corresponding to a given query. In this experiment, the retrieval score $r(C_i, A) \in [0, 1]$ of a result chunk C_i is defined as the percentage of words/tokens from the ground truth answer A of the given query that appear in that result chunk. For example, the ground truth answer (as collected in one quiz of the Cisco CCNA book) to the question "*Where does the Ethernet FCS field reside in?*" is "*It resides in the Ethernet trailer.*". Hence, a result chunk containing the words {*resides, Ethernet, trailer*} will have a score of 100% (stop words[3] and punctuation are removed), while one having only two of these words will have a score of 66.7%, and so on. For multi-answer questions, the score of a result chunk is given by the highest score among all computed ones related to each answer. Eventually, the final search score for a query $\overline{\boldsymbol{q}} \in Q_{\text{BERT}}$ (respectively $\overline{\boldsymbol{q}}' \in Q_{\text{NetBERT}}$) is then the maximum score among the computed top-k retrieval scores:

$$\begin{aligned} \text{score}_{\text{BERT}}(\overline{\boldsymbol{q}}) &= max_i r(C_i, A), \quad i \in S_{\text{BERT}} \\ \text{score}_{\text{NetBERT}}(\overline{\boldsymbol{q}}') &= max_{i'} r(C_{i'}, A), \quad i' \in S_{\text{NetBERT}}. \end{aligned} \tag{5.6}$$

Figure 5.12 summarizes the scoring approach of the information retrieval experiment.

Figure 5.12: Illustration of the scoring approach in the information retrieval experiment.

5.2.3 Results

Following the scoring procedure described previously, each query is given a search score according to the top-k result chunks retrieved from the similarity search based either on BERT embeddings (query $\overline{\boldsymbol{q}} \in Q_{\text{BERT}}$ compared against all text chunks $\overline{\boldsymbol{x}}_j \in X_{\text{BERT}}$ for $j = 1, ..., N$), or on NetBERT embeddings (query $\overline{\boldsymbol{q}}' \in Q_{\text{NetBERT}}$ compared against all text chunks $\overline{\boldsymbol{x}}'_j \in X_{\text{NetBERT}}$).

In this experiment, we arbitrarily chose $k = 5$ such that for each query, only the top-5 most similar chunks are considered in the computation of its final search score. Also, we investigated the three different distance/similarity functions discussed in the previous section to retrieve the k-closest vectors to the query vector (i.e., the *Euclidean* distance, the *dot-product* similarity and the *cosine* similarity). Table 5.10 shows the mean and median search scores (in percent) of BERT and NetBERT according to these functions.

It can be noticed that NetBERT largely outperforms BERT on this task of domain-specific information retrieval, with an average improvement between +10.6% and +12.3% depending on the distance/similarity measure used to compare the embeddings. Also, it seems that the choice of one or another distance/similarity function for comparing the embeddings has very little effect

[3]In computing, "stop words" are words which are filtered out before or after processing of natural language data (Rajaraman and Ullman, 2011), and usually refer to the most common words in a language (e.g., *the, is, at, which*). Here, we used the stop words list from the `nltk` library (`https://www.nltk.org/book/ch02.html#code-unusual`).

on the final search scores, except perhaps for the dot-product similarity which leads to a lower median score for both BERT and NetBERT. Given these very small differences, we arbitrarily chose to consider the Euclidean (L2) distance function for the rest of this experiment.

	BERT scores		NetBERT scores	
Dist./sim. function	Mean	Median	Mean	Median
Euclidean	59.4	66.7	70.5	**86.6**
Dot-product	58.3	61.3	**70.6**	83.3
Cosine	59.8	66.7	70.4	**86.6**

Table 5.10: Mean and median search scores (in percent) of BERT and NetBERT according to several distance/similarity functions used to retrieve the k-closest vectors to the query vector. The best results appear in **bold**.

Additionally, Figure 5.13 shows the histograms of the search score for both models when the Euclidean (L2) distance is used to compare the vectors. Interestingly, it can be seen that the number of queries that get a very low score (i.e., a score $\in [0, 10]$) is almost similar for both models (17 for BERT against 15 for NetBERT). However, the number of queries that get a very high score (i.e., a score $\in [90, 100]$) is much bigger for NetBERT (41 against 30 for BERT). Also, one can notice that the standard deviation is pretty much the same for the two models, and quite large in general. Therefore, a future investigation should be made with appropriate statistical testing to confirm with certainty that NetBERT performs indeed better than BERT on this task.

(a) BERT search scores.

(b) NetBERT search scores.

Figure 5.13: Histograms of the search scores for BERT and NetBERT when the Euclidean (L2) distance is used to compare the embeddings.

One should not forget that, in theory, a high score for a given query only means that most of the words from the corresponding ground truth answer appear in one of the top-k result chunks. It is then possible that these words coincidentally appear in the result chunks without completely or even partially answering the given question. Ideally, an expert in the field should rate each of the result chunks to a given question on a discrete scale according to whether or not that chunk answers the question. This would undoubtedly lead to a more accurate score, but in the same time would require to analyze $k \times m$ result chunks (m being the total number of queries and k the number of top results retrieved for each query), which is quite time expensive (e.g., there would be 450 chunks to analyze in this experiment). Instead, the scoring approach described in Section 5.2.2 allows to automate this rating, and turns out to be a pretty good approximate of the information retrieval ability in practice. Indeed, it has been noticed by analyzing some particular question-result pairs that in most cases, a high or perfect search score involves that one of the result chunks contains the right answer to the corresponding question. Some examples are shown in Table 5.11 for information retrieval using NetBERT embeddings.

Question-Result pairs	**Ground truth answers**
Q: Which protocols are examples of TCP/IP transport layer protocols? *R:* "The key difference between TCP and UDP is that TCP provides a wide variety of services to applications, whereas UDP does not. [...]"	UDP;TCP
Q: Which Ethernet standards defines Gigabit Ethernet over UTP cabling? *R:* "[...] For instance, the IEEE standardized Gigabit Ethernet support using inexpensive UTP cabling in standard 802.3ab. However, more often, engineers refer to that same standard as 1000BASE-T or simply Gigabit Ethernet. [...]"	1000BASE-T
Q: In the cabling for a leased line, what typically connects to a fourwire line provided by a telco? *R:* "[...] The four-wire cable from the telco plugs in to the CSU/DSU, typically using an RJ-48 connector [...]"	CSU;DSU
Q: Which Internet access technologies, used to connect a site to an ISP, offers asymmetric speeds? *R:* "[...] DSL uses the analog phone lines that are already installed in homes, while cable Internet uses the cable TV (CATV) cable."	DSL; Cable Internet
Q: Which routing protocols support VLSM? *R:* "[...] That second wave includes RIP Version 2 (RIPv2), OSPF Version 2 (OSPFv2) and Enhanced Interior Gateway Routing Protocol (EIGRP) [...]"	RIPv2;EIGRP;OSPF
Q: Which commands list the MAC address table entries for MAC addresses configured by port security? *R:* "[...] show mac address-table count Mac Entries for Vlan 1 [...]"	show mac address-table
Q: What command enables you to show the UDI of your Cisco router? *R:* "[...] show license udi Displays the UDI of the router. [...]"	show license udi

Table 5.11: Examples of question-result pairs retrieved with a similarity search based on NetBERT embeddings. The results (R) shown in the table are all top-k results retrieved with NetBERT embeddings using the L2 distance.

5.2.4 Further Analysis

In order to thoroughly analyze the differences between BERT and NetBERT embeddings applied to the task of domain-specific information retrieval, we investigated which result chunk among the top-k ones specifically led to the highest search score most of the time. The results are illustrated in Figure 5.14.

When performing the similarity-based search with NetBERT embeddings, it appears that it is mostly the first result chunk (i.e., the one that has the lowest L2 distance between its embedding and the query vector out of the top-k result chunks) that leads to the highest search

score. This is not the case when the search is performed with BERT embeddings. Indeed, the highest search score is then mostly achieved by both the first and second result chunks in equal proportion. Moreover, it can be noticed that with NetBERT, the percentage of the i-th result chunks (among the top-k ones) leading to the highest search score decreases as index i increases. In other words, the highest search score is mostly achieved with the first result (for 48% of the queries), then with the second one (for 22.7% of the queries), and so on. On the contrary, these percentages are much more evenly distributed across the different i-th result chunks when the search is performed with BERT (28.8%, 28.8% and 20.5% of the queries for the first, second and third result, respectively).

(a) BERT results.

(b) NetBERT results.

Figure 5.14: Distribution among the top-k result chunks leading to the highest search score.

In order to understand why the first result chunk retrieved with NetBERT embeddings leads more often to the highest search score among the top-k ones than the one retrieved with BERT embeddings, we computed the Euclidean (L2) distances between the embeddings of the two result chunks that have the highest search scores for each query. Intuitively, the larger this distance, the more we can consider that the first best result chunk better corresponds to the given query than the second one, and deserves its place at the top of the results list. Inversely, a small L2 distance between the embeddings of these chunks implies that the two results are roughly equivalent and that their order of appearance comes down to very little. The results are shown in Figure 5.15.

The main observation to be taken from these graphs is that with NetBERT, the median L2 distance between the embeddings of the best and second best result chunks is the largest when the best result chunk is the first result ($i = 1$ out of the top-k results). In contrast, it can be seen that with BERT, the median L2 distances are much closer to each other whatever the index i of the best result, and the highest median L2 distance is not even achieved when the best result is the first one, but well the fifth one ($i = 5$ out of the top-k results).

In brief, it was shown that the order of appearance of NetBERT results is more accurately related to the search score than it is with BERT results. In particular, the first result retrieved with NetBERT embeddings often achieves the highest search score among the top-k ones. Moreover, the L2 distance between the NetBERT embeddings of the first and second best results according to the search score is often larger when the best result is the first result ($i = 1$ out of the top-k results), which intuitively justifies its place at the top of the results list.

(a) BERT results.

(b) NetBERT results.

Figure 5.15: Euclidean (L2) distances between the embeddings of the two best result chunks according to the search score.

5.2.5 Conclusions

In this experiment, the quality of both BERT and NetBERT embeddings were evaluated on a task of domain-specific information retrieval based on a simple similarity search. To this end, a set of 90 questions related to computer networking were extracted from the Cisco CCNA certification book. The purpose of the experiment was to evaluate the ability of both models to retrieve the correct answer to a given question within the book.

The results showed that NetBERT is better at retrieving in-domain text chunks containing the right answers to the queries (average search score of 70.5% compared to 59.4% with BERT). In addition, it turns out that the order of appearance of the result chunks is more accurate with NetBERT than with BERT (e.g., the highest search score was achieved with the first result for 48% of the queries with NetBERT compared to only 28.8% with BERT).

In conclusion, it is clear that NetBERT captures a better representation of both domain-specific queries and text chunks so that the search results retrieved with a simple similarity-based operation are not only more accurate than those from BERT, but also appear in a more adequate order than with BERT.

5.3 Word Similarity

This experiment aims at comparing the quality of both BERT and NetBERT embeddings on the task of domain-specific *word similarity*. In the following, Section 5.3.1 first describes the task of word semantic similarity as an intrinsic evaluation method for word embedding models. Then, Section 5.3.2 discusses and compares the results of both BERT and NetBERT on the task for a few examples. Finally, Section 5.3.3 concludes the experiment by drawing a conclusion from the observed results.

5.3.1 Task Description

The goal of a word similarity evaluator is to measure how well the notion of human perceived similarity is captured in the vector-space word representations. Intuitively, word vectors that can capture word similarity might be expected to perform well on tasks that require a notion of explicit semantic similarity between words like paraphrasing or entailment. One of the most popular measures of semantic similarity in NLP is the *cosine* similarity, defined by

$$\cos(\boldsymbol{w}_x, \boldsymbol{w}_y) = \frac{\boldsymbol{w}_x \cdot \boldsymbol{w}_y}{\|\boldsymbol{w}_x\| \, \|\boldsymbol{w}_y\|}, \tag{5.7}$$

where $\boldsymbol{w}_x, \boldsymbol{w}_y \in \mathbb{R}^d$ are two d-dimensional word vectors and $\|\boldsymbol{w}_i\|$ $(i \in \{x, y\})$ indicate the l_2 norm of the word vectors. The resulting similarity denotes the cosine of the angle between the two vectors and therefore takes values in the interval $[-1, 1]$. The main advantages of this similarity measure is its low complexity, its robustness to scaling (thanks to the normalization operation) and its judgement on orientation rather than magnitude (unlike the L2 distance for example) which proved to be effective for comparing word embeddings computed with language models (Mikolov et al., 2013c; Zhang et al., 2019b).

5.3.2 Evaluations

In the case of contextual word embeddings (such as those output by BERT and NetBERT), each word from a given sentence is given a representation that depends on the context of that sentence. Hence, by considering a few sentences in which a same word is used with different meanings (so-called "homonymous" word) and by computing the *cosine similarity* between the embeddings of the different homonyms, we can analyze how well these embeddings capture semantic similarities. Since our focus is on computer networking, we considered three homonymous words each representing a popular concept in the field: a *bridge*, a *switch* and a *port*. Their different meanings are given in Table 5.12.

Word	Word class	Domain	Definitions
bridge	noun	General	Structure that is built over a river, road, or railway to allow people and vehicles to cross from one side to the other.
		Networking	Computer networking device that creates a single aggregate network from multiple communication networks.
switch	noun	General	Sudden or complete change.
		Networking	Device in a computer network that connects other devices together.
port	noun	General	Area of water and the land surrounding it where ships can take on and off goods and passengers.
		Networking	Communication endpoint in a computer network.

Table 5.12: Definitions of the words *bridge*, *switch* and *port* according to their general and networking meanings.

For each word, we arbitrarily chose three sentences in which the word is used with its general meaning, and three other sentences with its computer networking meaning.[4] These sentences are then fed into both BERT and NetBERT such that only the output representations of the word of interest (i.e., *bridge*, *switch* or *port*) are kept. Finally, the *cosine similarity* is computed for all combinations of embedding pairs output by each model. The results are shown in Figure 5.16, Figure 5.17 and Figure 5.18 for the words *bridge*, *switch* and *port*, respectively.

First of all, one can notice that the cosine values between all embeddings are quite high in general. However, this seems to be usual with BERT-based embeddings.[5] Hence, Xiao (2018) suggested to focus on the rank instead of the absolute *cosine* values. More precisely, he recommended that, given three words a, b, c and their corresponding d-dimensional word vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^d$, one should not conclude from $\cos(\boldsymbol{a}, \boldsymbol{b}) > th$ (e.g., $th = 0.9$) that $\boldsymbol{a}$ and $\boldsymbol{b}$ are similar, but instead should compare $\cos(\boldsymbol{a}, \boldsymbol{b})$ and $\cos(\boldsymbol{a}, \boldsymbol{c})$ and conclude that $\boldsymbol{a}$ is more similar to $\boldsymbol{b}$ than $\boldsymbol{c}$ if $\cos(\boldsymbol{a}, \boldsymbol{b}) > \cos(\boldsymbol{a}, \boldsymbol{c})$. That is why the *cosine* values were deliberately masked in the three figures.

In each example, the three first sentences use the word of interest with its general meaning, while the three last ones use it with its computer networking meaning. It appears that the *cosine*

[4] The general-context sentences were sampled from https://sentence.yourdictionary.com/, while the networking-context ones were sampled from the related Wikipedia page.

[5] https://github.com/hanxiao/bert-as-service#q-the-cosine-similarity-of-two-sentence-vectors-is-unreasonably-high-eg-always--08-whats-wrong

similarities between the embeddings of each of the words of interest are higher when the word is used with a similar meaning, and this with both BERT and NetBERT embeddings. Intuitively, it means that both BERT and NetBERT seem to capture the differences in meaning of each of the words in their representations.

By taking a closer look at each example individually, it can be noticed that NetBERT seems to better identify the similarity between the word *bridge* used in its networking context. Similarly, it appears that NetBERT better captures the similarity between the word *switch* but this time used in its general context (specifically for the third sentence of the example). However, it can be seen that BERT differentiates the meanings of the word *port* more clearly than NetBERT. These differences are very subtle though.

Figure 5.16: Visualisation of the cosine similarities between the embeddings of the word "*bridge*" used with its general and networking meanings.

Figure 5.17: Visualisation of the cosine similarities between the embeddings of the word "*switch*" used with its general and networking meanings.

(a) BERT results. (b) NetBERT results.

Figure 5.18: Visualisation of the cosine similarities between the embeddings of the word "*port*" used with its general and networking meanings.

5.3.3 Conclusions

In conclusion, this word similarity evaluation between BERT and NetBERT embeddings reveals that both models manage to capture differences in meaning in their representations. From the few examples presented in this experiment, it appears that NetBERT performs similarly to (or slightly improves over) BERT in its ability to identify similarities between homonymous words with a computer networking meaning when used in that context. However, the differences are very subtle and more evidence would be needed to draw a general conclusion.

5.4 Word Analogy

This experiment aims at comparing the quality of both BERT and NetBERT embeddings on the task of domain-specific *word analogy*. In the following, Section 5.4.1 first describes the task of word analogy as an intrinsic word embeddings evaluation method. Then, Section 5.4.2 discusses and compares the results of both BERT and NetBERT on the task for a few examples. Finally, Section 5.4.3 ends the experiment by summarizing the conclusions that can be drawn from it.

5.4.1 Task Description

Word analogy is the second most popular way (after *word similarity*) to determine if the word embeddings from a given language model capture some syntactic or semantic relationships between the words. Given two pairs of words (a, a') and (b, b'), the analogy relationship between these two is usually expressed as

$$a : a' :: b : b', \tag{5.8}$$

and it reads « a is to a' as b is to b' ». In practice, an analogy measure $m \in \mathbb{R}$ can be computed for such a relationship so that

$$m = s(\boldsymbol{x}_{b'}, \boldsymbol{x}_{a'} - \boldsymbol{x}_a + \boldsymbol{x}_b), \tag{5.9}$$

where the $\boldsymbol{x}_i \in \mathbb{R}^d$ $(i \in \{a, a', b, b'\})$ represent the d-dimensional word representations and $s : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a similarity function (typically the *cosine* similarity). Hence, a high similarity means that the vector pairs share a similar direction.

Alternatively, such relationships can be visualised by plotting the vector offsets between the dimensionally-reduced word embeddings of the different pairs of words. The most commonly used strategy for word embeddings dimensionality reduction is t-Stochastic Neighbor Embedding (t-SNE; Maaten and Hinton, 2008), as used and discussed in Section 5.1.4. However, this method does not lend itself well to the analysis of word analogies. Indeed, t-SNE is usually used to provide a high-level overview of the embedding space through non-linear dimensionality reduction. Although such a space can reveal some interesting separations between word groups (e.g., countries, nouns, verbs, etc.), they inherently distort the linear (i.e., semantic) relationships most interesting to our study. Consequently, to preserve such relationships, linear projections are preferred. The most common approach is then to use Principal Component Analysis (PCA; Pearson, 1901) restricted to carefully chosen subsets of words.

5.4.2 Evaluations

Unlike Word2Vec, BERT and NetBERT embeddings rely heavily on contextual information, as both models were specifically pre-trained such that context is used to build the word representations. That is why in this experiment, the words are not embedded as they are, without any context, but are instead put into short sentences that use them in their appropriate contexts. These sentences are then passed as inputs to both BERT and NetBERT, and only the word representations of the four words of interest (i.e., the words from the given analogy) are gathered, dimensionally-reduced with PCA and plotted in two dimensions.

Arguably, the most famous example of word analogies in the NLP community is the «*man* : *woman* :: *king* : *queen*» analogy from Mikolov et al. (2013c), who showed that the vector offsets between the dimensionally-reduced Word2vec representations of the pairs (*man*, *woman*) and (*king*, *queen*) were equivalent in the reduced dimensional space, highlighting the expected male/female relationship. Here, we began by replicating this example with BERT and NetBERT embeddings. The aim was firstly to check if BERT learned to capture semantic relationships in a similar way as Word2vec, and secondly to analyze if such general-domain relationships were also captured (i.e., not lost) with NetBERT embeddings. The results are shown in Figure 5.19.

(a) BERT embeddings. (b) NetBERT embeddings.

Figure 5.19: Comparison of vector offsets between the dimensionally-reduced BERT and NetBERT embeddings for two word pairs illustrating the *gender* relationship.

At first glance, both BERT and NetBERT embeddings seem to capture the gender relationship between the pairs of words, as the vector offsets are almost perfectly parallel. It does seem, however, that the relationship is somewhat more accurately defined with BERT as the vector offsets are much closer in length than those from NetBERT.

Now, we would like to investigate if NetBERT is able to capture some relationships between computer networking concepts. Finding human perceived analogies is a difficult and very subjective task though, even more for a technical field such as computer networking. Therefore, several networking experts were asked to enumerate a few analogies between computer networking concepts to conduct this experiment. As for the «*man* : *woman* :: *king* : *queen*» example, each word was placed in its appropriate computer networking context for embedding. Figure 5.20, Figure 5.21 and Figure 5.22 illustrate the vector offsets between the dimensionally-reduced BERT and NetBERT embeddings of several pairs of words representing the *protocol type*, *address type* and *layer type* relationships, respectively.

In general, it seems that NetBERT capture the relationships between the different networking concepts better than BERT. Indeed, unlike BERT, the vectors offsets between the dimensionally-reduced NetBERT embeddings are almost equivalent for each example, highlighting the expected relationships.

(a) BERT embeddings. (b) NetBERT embeddings.

Figure 5.20: Comparison of vector offsets between the dimensionally-reduced BERT and NetBERT embeddings for two word pairs illustrating the *protocol type* relationship.

(a) BERT embeddings. (b) NetBERT embeddings.

Figure 5.21: Comparison of vector offsets between the dimensionally-reduced BERT and NetBERT embeddings for two word pairs illustrating the *address type* relationship.

(a) BERT embeddings.

(b) NetBERT embeddings.

Figure 5.22: Comparison of vector offsets between the dimensionally-reduced BERT and NetBERT embeddings for two word pairs illustrating the *layer type* relationship.

5.4.3 Conclusions

In brief, this small investigation tends to show the ability of NetBERT to capture semantic relationships between networking concepts, unlike BERT. However, it is important to note that this experiment used a very small sample size (out of the vast lexicon of english words) and one should not overgeneralize with such limited evidence. Therefore, further experimentation on the subject is needed to draw any general conclusion.

Chapter 6

NetBERT Search Engine

This chapter covers a practical application for NetBERT, motivated by the Cisco search problem discussed in Section 1.1. Specifically, it describes how NetBERT can be used to implement a simple similarity-based search engine on computer networking text corpora. First, Section 6.1 details the implementation of the search engine. Then, Section 6.2 reviews two networking text corpora on which the search engine retrieves information. Finally, Section 6.3 discusses the limitations of the current approach and some possible modifications to improve the system speed, its memory usage and the quality of its search results.

6.1 Implementation

In our search engine, NetBERT is used to build vector representations of pieces of text (also called "chunks"). Then, these representations are used to perform a nearest neighbor search to a given query. The implementation of our system is divided into two main steps, as shown in Figure 6.1: the *index creation* and the *real-time* search.

Figure 6.1: Illustration of the process of using NetBERT embeddings to build a similarity-based search engine.

6.1.1 Index Creation

Given a corpus of documents in which we intend to search for information, the first step consists in creating an index that will store a set of vectors representing text chunks from that corpus.

Initially, each document from the search corpus is split into groups of consecutive sentences (chunks) of a maximum length. This length is defined as the maximum number of input tokens that NetBERT is able to process, which is $T = 512$ tokens.

Then, the text chunks are fed into NetBERT. As a reminder, NetBERT outputs a vector representation $\boldsymbol{x}_i \in \mathbb{R}^d$ for each token i of an input sequence (where $d = 768$ is the output representation dimension of the model). In order to get a representation of the full sequence, the two most common embedding approaches are either to average the last layer token representations, or to use the `[CLS]` token embedding. It was recently shown (Reimers and Gurevych, 2019; Wang and Kuo, 2020) that the averaging approach captures a more inherent structure of the sentence while the `[CLS]` token representation is more suitable for downstream classification tasks. Following these conclusions, the chunk representations were computed by averaging the output token representations of NetBERT, resulting in a vector $\overline{\boldsymbol{x}} \in \mathbb{R}^d$ such that

$$\overline{\boldsymbol{x}} = \frac{1}{T} \sum_{i=1}^{T} \boldsymbol{x}_i. \tag{6.1}$$

Finally, the set of chunk embeddings is used to create an index data structure. Here, we rely on the Facebook AI Similarity Search (`Faiss`) library (Johnson et al., 2019) to implement the index. `Faiss` is a library developed by Facebook AI Research for efficient similarity search and clustering of dense vectors. It is built on a few basic algorithms such as k-means clustering (Lloyd, 1982), Principal Component Analysis (Pearson, 1901) and Product Quantization encoding/decoding (Jegou et al., 2010), with very efficient implementations written in C++ and complete wrappers for Python. In addition, the library offers an optional GPU implementation for these algorithms, providing what is likely the fastest exact and approximate (compressed-domain) nearest neighbor search implementation for high-dimensional vectors. Given the large computational resources at our disposal (i.e., 8×32GB NVIDIA Tesla V100 GPUs), we decided to create an index for *exact* k-nearest neighbor search, i.e., brute-force kNN without compression of the original vectors. Therefore, the flat index is simply a matrix $\boldsymbol{M} \in \mathbb{R}^{N \times d}$, where N is the number of chunks extracted from the corpus and $d = 768$ the output representation dimension of NetBERT.

6.1.2 Real-time Search

Once the index is created and loaded in memory, it is possible to search in real-time given a query and a pre-selected distance function. This function is used to compute a distance between the query and all chunk embeddings from the index.

Basically, the query is first embedded with NetBERT as a d-dimensional vector $\overline{\boldsymbol{q}} \in \mathbb{R}^d$, being the average of the token representations from that query,

$$\overline{\boldsymbol{q}} = \frac{1}{T} \sum_{i=1}^{T} \boldsymbol{q}_i. \tag{6.2}$$

Then, all the indexed vectors $\overline{\boldsymbol{x}}_j$ are compared to the query embedding $\overline{\boldsymbol{q}}$ with respect to the chosen distance function $f : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$. Eventually, the indices of the k closest vectors to the query representation are returned, i.e., the set

$$S = k\text{-argmin}_j f(\overline{\boldsymbol{q}}, \overline{\boldsymbol{x}}_j), \quad j = 1, ..., N. \tag{6.3}$$

Note that the distance function f is typically the *Euclidean* (L2) distance. Alternatively, a similarity function $s : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ can also be used, in which case the k nearest neighbors to the query $\overline{\boldsymbol{q}}$ are searched so that

$$S = k\text{-argmax}_j s(\overline{\boldsymbol{q}}, \overline{\boldsymbol{x}}_j), \quad j = 1, ..., N. \tag{6.4}$$

Commonly used similarity functions are the *dot-product* similarity and the *cosine* similarity.

Finally, a quick look at these indices in the saved text chunks database allows to retrieve the pieces of text corresponding to these k closest vectors, as shown on the right hand-side of Figure 6.1.

6.2 Applications

The similarity-based search engine described previously was implemented as an interactive `Jupyter` notebook, as shown in Figure 6.2. An index was created for two different networking corpora, namely the Cisco corpus and the RFC corpus.

Figure 6.2: The NetBERT search engine implemented as an interactive Jupyter notebook.

6.2.1 Cisco Corpus

The first index was created in order to search from the complete Cisco text corpus on which NetBERT was pre-trained (see Section 4.2). As a reminder, this corpus contains about 170M sentences retrieved from nearly 440K different web pages from `cisco.com`. After splitting this corpus into text chunks of maximum length $T = 512$ tokens, it resulted in an index of 25.7GB storing the representations of about 9M text chunks. A summary of these statistics is given in Table 6.1. At run-time, it takes about 0.05s to encode the query and 0.03s to perform the kNN search with `Faiss` on a single 32GB NVIDIA Tesla V100 GPU, resulting in a total search time of less than 0.1s.

6.2.2 RFC Corpus

The second index was created in order to retrieve information from Request for Comments (RFC) documents.[1] In information and communications technology, a RFC is a type of document authored by engineers and computer scientists in the form of a memorandum describing methods, behaviors, research, or innovations applicable to the working of the Internet and Internet-connected systems. Such documents may come from many bodies including from the Internet Engineering Task Force (IETF), the Internet Research Task Force (IRTF), the Internet Architecture Board (IAB), or from independent authors. Many RFCs are informational or experimental in nature and are submitted either for peer review or to convey new concepts. Occasionally, the IETF adopts some of the proposals published as RFCs as Internet Standards.

This type of documents lend themselves very well to our application as their vocabulary is mainly related to the field of computer networking. In addition, these documents are very easy to extract as they are all available online in `.txt` format. Hence, a total of 8,570 RFC documents were collected and split into about 1.3M text chunks. These chunks were then embedded with NetBERT to eventually create a `Faiss` index with a size of 3.7GB, as shown in Table 6.1. At run-time, it only takes about 0.01s to perform the similarity search with `Faiss` on a single 32GB NVIDIA Tesla V100 GPU.

[1] https://tools.ietf.org/rfc/index

Corpus	Number of documents	Number of text chunks	Size of `Faiss` index
`cisco.com`	442K	9M	25.7GB
RFCs	8.6K	1.3M	3.7GB

Table 6.1: Statistics about the text corpora used with the NetBERT search engine.

6.3 Limitations and Possible Improvements

Although sometimes returning fairly query-related results, this simple search engine is currently far from being perfect. One of the main drawbacks of the current system is that it is highly sensitive to the syntactic form of the query. For example, when the query appears as a well-phrased question, the results will most likely be similar questions about the same or a related topic. Upon reflection, this problem is probably related to the way sentences are represented. For the moment, the representation of a sentence (or a chunk) is simply the average of the token representations output by NetBERT for that sentence. More effective ways of computing such vectors are discussed in the following. In addition, we discuss some methods to improve the speed and memory usage of the current search engine system.

6.3.1 Sentence Representation

Sentence embedding is an important research topic in NLP. While BERT achieves state-of-the-art performance in quite a few NLP tasks using contextual word representations, generating high quality sentence representations from BERT-based word models is an open problem. Lately, there has been some research addressing this particular issue. For example, Reimers and Gurevych (2019) studied the performances of commonly used sentence representations methods such as Universal Sentence Encoder (Cer et al., 2018), InferSent (Conneau et al., 2017) and average GloVe (Pennington et al., 2014) embeddings, and compared them with average BERT embeddings and BERT's `[CLS]` token representation on various semantic textual similarity (STS) tasks. Eventually, they showed that the direct use of BERT outputs for sentence representation generates rather poor performance. Averaging BERT outputs led to better results than using the `[CLS]` token representation, but both methods were actually worse than a static word embedding scheme such as averaging GloVe embeddings on the tested tasks. To overcome this issue, they introduced Sentence-BERT (SBERT), a modification of the pre-trained BERT network that uses siamese and triplet networks (Schroff et al., 2015) to update the weights with a fine-tuning operation such that the produced sentence embeddings are semantically meaningful. They showed that their approach substantially improves the performance on STS, outperforming both InferSent and Universal Sentence Encoder.

Recently, Wang and Kuo (2020) confirmed the conclusions drawn by Reimers and Gurevych. They hypothesize that the poor performance of BERT outputs directly used for sentence representation could be attributed to the fact that the model is not trained using a similar objective function, in the sense that masked language model (MLM) and next sentence prediction (NSP) objectives are not suitable for a linear integration of representations. Therefore, they proposed a new sentence embedding method by dissecting BERT-based word models through geometric analysis of the space spanned by the word representation. Their novel method, called SBERT-WK, doesn't require further fine-tuning and outperforms SBERT on a range of supervised downstream tasks.

In conclusion, such novel methods could further be used in our search engine system to improve the quality of the sentence/chunk embeddings.

6.3.2 Speed and Memory

The current search engine system uses brute-force search on the original chunk embeddings. If speed or memory were to become a problem in the future, two options can be considered either separately or implemented together.

First, the `Faiss` library allows to use a compressed representation of the original vectors based on product quantizers (Jegou et al., 2010), which can save a lot of memory. This generally comes at the cost of a less precise search but these methods can scale to billions of vectors in main memory on a single server. Basically, indexes based on such compression methods just encode the vectors into codes of a fixed size c and store them in a compressed matrix $\boldsymbol{M'} \in \mathbb{R}^{N \times c}$, where N is the number of chunks extracted from the corpus. At search time, all the indexed vectors are either decoded sequentially and compared to the query vectors, or directly compared to the query in the compressed domain, which is faster.

To speed up the search, it is also possible to use an index that segments the original dataset of chunk embeddings into pieces, what `Faiss`' authors call Voronoi cells (Voronoi, 1908), such that each d-dimensional embedding falls in one of these cells. This type of index requires a training stage in order to create the clusters (typically using the k-means algorithm). Once created, each cluster is then represented by its centroid vector. At search time, the query vector is first compared to the set of centroids to determine in which cluster it falls in. Then, only the chunk embeddings contained in that cluster and a few neighboring ones are compared against the query vector for final results.

Chapter 7

Conclusions

To the best of our knowledge, this thesis corresponds to the first attempt in pre-training a large Transformer-based language representation model on *computer networking* text corpora. All the information that has been presented in this work aimed to investigate if our novel model, called NetBERT, could improve performance on *computer networking* mining tasks over a similar model pre-trained exclusively on general-domain text, namely BERT (Devlin et al., 2018). Based on several quantitative and qualitative studies, it can be concluded that our novel model, resulting from an extensive pre-training of about a month on 8×32GB NVIDIA Tesla V100 GPUs, indeed outperforms the general-domain model when evaluated on *domain-specific* language tasks. The results indicate that an additional pre-training on *computer networking* text helps the model to understand more accurately domain-related sentences and conceptual relationships.

To arrive at this conclusion, the performance of both BERT and NetBERT was extrinsically and intrinsically evaluated on various language tasks in Chapter 5. The results showed that NetBERT outperforms BERT on the task of domain-specific *text classification*, with an improvement of +0.9% F1 and +1.3% MCC (Matthews correlation coefficient). However, the experiment also proved that, although not having been pre-trained on computer networking text, BERT still does a very good job at classifying domain-specific sentences. Then, both models were evaluated and compared on the task of in-domain *information retrieval*. This time, NetBERT showed significant improvements over BERT regarding a custom search score (+12.3% improvement). Further analysis also proved that the order of appearance of the retrieval results was more accurate with NetBERT than it was with BERT. Furthermore, an evaluation on *word similarity* between a few homonymous words representing a popular networking concept tended to show that NetBERT performs slightly better than (or similarly to) BERT in its ability to capture the semantic meaning of some networking words. Lastly, a *word analogy* evaluation between some computer networking concepts reveals that NetBERT is able to capture semantic relationships between these concepts, unlike BERT. Overall, these results give a clear answer to the main research question of this thesis, which was: "*Does a large language representation model pre-trained on domain-specific text improve performance over the same model pre-trained on general-domain corpora, when evaluated on domain-specific text mining tasks?*".

In addition to that main research question, we managed to provide insightful answers to the secondary questions that have been presented at the beginning of this work in Chapter 1. First, an answer to the question "*Which language representation model should be considered for the purpose of domain-specific text mining given limited computational resources?*" was provided in Chapter 4. Through a discussion on existing state-of-the-art language representation models, particularly focused on the size of these models (which is the main limitation of pre-training Transformer-based models), we eventually concluded that the base version of BERT was the most suitable model according to its performance/size balance. That choice was also motivated by the recent integration of BERT in the Google Search engine as well as its increasing use for many NLP problems such as domain-specific text mining tasks and factual knowledge bases.

The second question, "*What are some good pre-training strategies to consider in order to achieve the best possible model with one unique run given the huge computational time related to the pre-training of current state-of-the-art language representation models?*", was answered in Chapter 4. Through an extensive review of the literature about BERT-based models, we summarized some of the best pre-training strategies for such models regarding the use of the next sentence prediction (NSP) objective, a custom vs. general vocabulary, a cased vs. uncased vocabulary, a static vs. dynamic masking process, and a scratch vs. checkpoint pre-training.

Finally, the third and most important secondary research question was: "*How can language representation models be efficiently evaluated on domain-specific tasks, given little or no labelled data?*". Most of the time, language representation models are extrinsically evaluated on one or several downstream NLP tasks using either a feature-based approach or a fine-tuning approach. The performance of the model is then measured on a labelled dataset for the given NLP task, and is usually considered as a measure of word embedding quality. In this thesis, we only had a single labelled dataset at hand for the task of *text classification*. Therefore, we used the fine-tuning approach and compared the performance of both models with relevant classification metrics, as presented in the first experiment of Chapter 5. In order to advance further proofs of the better performance of NetBERT over BERT, we came with a novel carefully crafted extrinsic evaluation built on a feature-based approach for domain-specific *information retrieval*. Additionally, we presented two intrinsic evaluations that aimed to evaluate the embedding quality of both models by assessing with human judgments the semantic properties and relationships captured within the embeddings.

Despite these contributions, there is still space for future research to complement the work that has been presented in this thesis. Some potential interesting ideas for future directions are presented in the next section.

7.1 Future Directions

From a research perspective, we have shown that NetBERT improves performance over BERT when evaluated on a few language tasks related to computer networking. However, we are aware that further evaluations are needed to bring more evidence to the conclusion that is drawn in this thesis. Although *intrinsic* evaluations might reveal some interesting insights of word embeddings, they are sometimes criticized as they mainly rely on human assessments that might sometimes be biased by certain subjective factors. For that reason, we believe that practitioners should particularly focus on additional *extrinsic* experiments for other common downstream NLP tasks. At the time of writing, the Cisco One Search team from San Jose, California, is creating a novel dataset in the SQuAD (Rajpurkar et al., 2016) format for computer networking question-answering (QA), as well as a networking dataset for semantic textual similarity (STS). Future evaluations on these two datasets would definitively allow to draw more general conclusions when combined to the results presented in this thesis.

From a business perspective, we believe that NetBERT could be used for many useful applications at Cisco. One key example, which was discussed in Chapter 1 and initially motivated the work done in this thesis, is its use for information retrieval in order to improve the poor search results of the current Cisco search engine. In Chapter 6, we introduced a functional proof of concept (POC) of a simple search engine that uses a similarity-based approach with NetBERT embeddings for networking information retrieval. We believe that this POC can be used as a baseline for a novel search engine either based exclusively on NetBERT embeddings, or on a combination of traditional information retrieval methods such as TF-IDF (Jones, 1972) or BM25 (Robertson et al., 1995) to get a first set of candidate results, and NetBERT embeddings to further re-rank these results. However, future work should definitely investigate more accurate ways of representing sentences from the word embeddings output by NetBERT, as mentioned at the end of Chapter 6.

This thesis ends with the hope of having provided the reader with insightful knowledge about the use of a domain-specific language representation model for networking-related text mining tasks. In addition, we hope that this work can become a starting point for further research on the topic or potential business applications based on the NetBERT model.

Appendix A

Additional Details about BERT

A.1 Model Parameters

BERT-base (L=12, h=12, d_{model}=768)				
Layer	**Sublayer**	**Weight matrices**	**Dimensionality**	**#Parameters**
Input	Token	$\boldsymbol{W}^{TE}$	$[30522 \times 768]$	23,440,648
	Position	$\boldsymbol{W}^{PE}$	$[512 \times 768]$	393,216
	Segment	$\boldsymbol{W}^{SE}$	$[2 \times 768]$	1,536
Encoder #1	Self-Attention	$\boldsymbol{W}^{Q}, \boldsymbol{W}^{K}, \boldsymbol{W}^{V}$	$[768 \times 64] \cdot 3$	147,456
		$\boldsymbol{P}_i^{Q}, \boldsymbol{P}_i^{K}, \boldsymbol{P}_i^{V}$	$([768 \times 64] \cdot 3) \cdot 12$	1,769,472
		$\boldsymbol{W}^{O}$	$[768 \times 768]$	589,824
	Feed-Forward	$\boldsymbol{W}_1, \boldsymbol{b}_1$	$[768 \times 3072], [3072 \times 1]$	2,362,368
		$\boldsymbol{W}_2, \boldsymbol{b}_2$	$[3072 \times 768], [768 \times 1]$	2,360,064
	Residual	$\boldsymbol{W}^{R}$	$[768 \times 1] \cdot 2$	1,536
...	...		...	...
Encoder #12	Self-Attention	$\boldsymbol{W}^{Q}, \boldsymbol{W}^{K}, \boldsymbol{W}^{V}$	$[768 \times 64] \cdot 3$	147,456
		$\boldsymbol{P}_i^{Q}, \boldsymbol{P}_i^{K}, \boldsymbol{P}_i^{V}$	$([768 \times 64] \cdot 3) \cdot 12$	1,769,472
		$\boldsymbol{W}^{O}$	$[768 \times 768]$	589,824
	Feed-Forward	$\boldsymbol{W}_1, \boldsymbol{b}_1$	$[768 \times 3072], [3072 \times 1]$	2,362,368
		$\boldsymbol{W}_2, \boldsymbol{b}_2$	$[3072 \times 768], [768 \times 1]$	2,360,064
	Residual	$\boldsymbol{W}^{R}$	$[768 \times 1] \cdot 2$	1,536
Total number of parameters				110,604,288

Table A.1: **Model parameters of BERT-base**. L is the number of layers, h the number of attention heads, and d_{model} the dimensionality of the input/output in the model.

BERT-large (L=24, h=16, d_{model}=1024)				
Layer	**Sublayer**	**Weight matrices**	**Dimensionality**	**#Parameters**
Input	Token	$\boldsymbol{W}^{TE}$	$[30522 \times 1024]$	31,254,528
	Position	$\boldsymbol{W}^{PE}$	$[512 \times 1024]$	524,288
	Segment	$\boldsymbol{W}^{SE}$	$[2 \times 1024]$	2,048
Encoder #1	Self-Attention	$\boldsymbol{W}^{Q}, \boldsymbol{W}^{K}, \boldsymbol{W}^{V}$	$[1024 \times 64] \cdot 3$	196,608
		$\boldsymbol{P}_i^{Q}, \boldsymbol{P}_i^{K}, \boldsymbol{P}_i^{V}$	$([1024 \times 64] \cdot 3) \cdot 16$	3,145,728
		$\boldsymbol{W}^{O}$	$[1024 \times 1024]$	1,048,576
	Feed-Forward	$\boldsymbol{W}_1, \boldsymbol{b}_1$	$[1024 \times 4096], [4096 \times 1]$	4,198,400
		$\boldsymbol{W}_2, \boldsymbol{b}_2$	$[4096 \times 1024], [1024 \times 1]$	4,195,328
	Residual	$\boldsymbol{W}^{R}$	$[1024 \times 1] \cdot 2$	2,048
...	...		...	...
Encoder #24	Self-Attention	$\boldsymbol{W}^{Q}, \boldsymbol{W}^{K}, \boldsymbol{W}^{V}$	$[1024 \times 64] \cdot 3$	196,608
		$\boldsymbol{P}_i^{Q}, \boldsymbol{P}_i^{K}, \boldsymbol{P}_i^{V}$	$([1024 \times 64] \cdot 3) \cdot 16$	3,145,728
		$\boldsymbol{W}^{O}$	$[1024 \times 1024]$	1,048,576
	Feed-Forward	$\boldsymbol{W}_1, \boldsymbol{b}_1$	$[1024 \times 4096], [4096 \times 1]$	4,198,400
		$\boldsymbol{W}_2, \boldsymbol{b}_2$	$[4096 \times 1024], [1024 \times 1]$	4,195,328
	Residual	$\boldsymbol{W}^{R}$	$[1024 \times 1] \cdot 2$	2,048
Total number of parameters				338,661,376

Table A.2: **Model parameters of BERT-large**. L is the number of layers, h the number of attention heads, and d_{model} the dimensionality of the input/output in the model.

A.2 Model Architecture

Figure A.1: 3D visualisation of the BERT-base architecture.[1]

[1] https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/bert-encoder

A.3 BERT in Google Search

(a) **Example of a search for "2019 brazil traveler to usa need a visa"**. Here, the word "to" and its relationship to the other words in the query are particularly important to understanding the meaning of the query. It is about a Brazilian traveling to the U.S., and not the other way around. With BERT, Search is now able to grasp this nuance, providing a much more relevant result for this query.

(b) **Example of a search for "do estheticians stand a lot at work"**. Previously, Google Search was taking an approach of matching keywords, matching the term "stand-alone" in the result with the word "stand" in the query. With BERT, Search now understands that "stand" is related to the concept of the physical demands of a job, and displays a more appropriate response.

Figure A.2: Demonstration of BERT's ability to understand the intent behind a search query.[2]

[2] https://www.blog.google/products/search/search-language-understanding-bert/

Appendix B

Additional Resources about the Experiments

B.1 Text Classification

B.1.1 Classification Performance Metrics

In order to define some commonly used classification metrics, let us first remind the notions of True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) samples in a binary classification problem:

- TP are samples that were classified positive and are really positive.
- TN are samples that were classified negative and are really negative.
- FP are samples that were classified positive but should have been classified negative.
- FN are samples that were classified negative but should have been classified positive.

Here, TP, TN, FP and FN stand for the respective number of samples in each of the classes. Let us now define accordingly some popular binary classification metrics.

Precision The *precision* (P) is defined as

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{B.1}$$

Recall The *recall* (R) is defined as

$$\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{B.2}$$

F1-score The *F1-score* (F1) is defined as the weighted average between precision and recall,

$$\mathrm{F1} = 2\left(\frac{\mathrm{P} \times \mathrm{R}}{\mathrm{P} + \mathrm{R}}\right). \tag{B.3}$$

In order to compute these three performance metrics in a multi-class classification, two common choices are *macro-averaging* and *weighted-averaging*. Macro-averaging is simply an arithmetic mean of the per-class metrics,

$$x_M = \frac{1}{N}\sum_{i=1}^{N} x_i, \tag{B.4}$$

where N is the number of classes in the classification problem. With that approach, each class is given an equal weight. In contrast, weighted-averaging takes class imbalance into account by weighting the different metrics x_i of each class by the number of samples w_i from that class,

$$x_W = \frac{\sum_{i=1}^{N} w_i x_i}{\sum_{i=1}^{N} w_i}. \tag{B.5}$$

Matthews Correlation Coefficient The *Matthews Correlation Coefficient* (MCC) is defined as

$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}. \tag{B.6}$$

The MCC is said to be more informative than the F1-score in evaluating binary classification problems (Chicco and Jurman, 2020), as it takes into account the balance ratios of the four confusion matrix categories (TP, TN, FP and FN). It has also been generalized to the multi-class case, where it is defined in terms of a confusion matrix $C^{K\times K}$ (K being the number of classes) such that

$$\mathrm{MCC} = \frac{\sum_k \sum_l \sum_m C_{kk} C_{lm} - C_{kl} C_{mk}}{\sqrt{\sum_k \left(\sum_l C_{kl}\right) \left(\sum_{k'|k'\neq k} \sum_{l'} C_{k'l'}\right)} \sqrt{\sum_k \left(\sum_l C_{lk}\right) \left(\sum_{k'|k'\neq k} \sum_{l'} C_{l'k'}\right)}}. \tag{B.7}$$

B.1.2 Optimal Hyperparameters Search

Batch size	Learning rate	Number of epochs	BERT			NetBERT		
			P	R	F1	P	R	F1
16	5e-5	1	88.94	88.91	88.9	89.49	88.77	88.95
		2	91.31	91.14	91.2	92.27	92.1	92.15
		3	92.3	92.28	92.28	93.44	93.49	93.46
		4	92.63	92.72	92.63	93.55	93.53	93.53
		5	93.27	93.26	93.24	94.36	94.36	94.36
		6	93.54	93.55	93.52	94.29	94.3	94.29
	3e-5	1	89.16	89.14	89.13	90.46	90.2	90.26
		2	92.05	91.91	91.96	92.49	92.28	92.36
		3	92.41	92.43	92.38	93.52	93.47	93.49
		4	93.07	93.12	93.06	93.35	93.45	93.37
		5	93.25	93.22	93.2	94.32	94.3	94.31
		6	**93.63**	**93.64**	**93.62**	**94.46**	**94.47**	**94.46**
	2e-5	1	88.69	88.5	88.57	89.73	89.93	89.79
		2	91.14	90.87	90.94	92.39	92.26	92.31
		3	92.19	92.14	92.14	93.09	93.09	93.09
		4	92.98	92.95	92.93	93.57	93.64	93.59
		5	93.1	93.07	93.09	93.78	93.76	93.76
		6	93.16	93.14	93.13	94.28	94.3	94.29
32	5e-5	1	88.69	88.5	88.58	90.07	89.89	89.91
		2	91.76	91.49	91.58	92.21	91.93	92.01
		3	92.5	92.51	92.5	93.23	93.24	93.2
		4	93.02	93.01	92.99	93.12	93.22	93.14
		5	93.2	93.22	93.2	93.9	93.93	93.9
		6	93.58	93.59	93.58	94.43	**94.47**	94.44
	3e-5	1	88.9	88.85	88.85	90.57	90.58	90.55
		2	91.26	90.83	90.95	92.09	91.85	91.92
		3	92.24	92.14	92.15	93.13	93.16	93.1
		4	92.91	92.91	92.9	93.3	93.32	93.29
		5	93.42	93.41	93.38	93.77	93.78	93.76
		6	93.35	93.3	93.31	94.19	94.22	94.19
	2e-5	1	88.32	88.1	88.18	90.04	90.14	90.08
		2	90.93	90.77	90.82	92.2	91.99	92.04
		3	92.27	92.26	92.25	93.09	93.16	93.11
		4	92.43	92.45	92.41	93.34	93.39	93.35
		5	92.62	92.62	92.58	93.66	93.72	93.67
		6	92.92	92.91	92.9	93.53	93.53	93.54

Table B.1: Results of BERT and NetBERT on the validation set of the query classification dataset. The reported metrics are the *weighted-average* Precision (P), Recall (R) and F1-score (F1), all expressed in percent. The best scores appear in **bold**, and the second best scores are underlined.

Batch size	Learning rate	Number of epochs	BERT			NetBERT		
			P	R	F1	P	R	F1
16	5e-5	1	84.99	83.96	84.42	84.65	84.84	84.41
		2	88.21	87.21	87.65	89.52	88.68	89.03
		3	88.45	88.48	88.43	91.15	89.84	90.48
		4	91.17	88.1	89.57	91.46	90.36	90.89
		5	90.54	90.47	0.9047	92.24	91.81	92.01
		6	91.31	90.27	90.75	91.95	91.74	91.84
	3e-5	1	86.2	84.17	85.1	87.3	84.81	85.91
		2	89.04	88.36	88.64	89.3	89.66	89.43
		3	89.11	88.85	88.92	90.04	90.98	90.5
		4	91.04	88.82	89.88	92.06	89.48	90.73
		5	90.7	90.01	90.28	91.94	91.74	91.83
		6	91.38	90.46	90.89	92.46	**91.83**	92.14
	2e-5	1	84.01	83.41	83.66	86.86	83.68	85.21
		2	88.12	87.04	87.43	88.84	89.37	89.07
		3	88.4	89.16	88.74	90.32	90.08	90.2
		4	90.74	88.96	89.77	91.99	90.08	91.01
		5	90.01	90.21	90.1	91.17	90.95	91.05
		6	90.38	89.76	90.03	92.14	91.65	91.89
32	5e-5	1	83.3	84.16	83.71	86.07	85.61	85.71
		2	88.29	88.25	88.16	88.02	90.19	89.01
		3	89.07	89.48	89.26	91.66	89.82	90.66
		4	91.05	89.11	90.02	91.61	89.1	90.32
		5	90.69	90.04	90.34	91.97	91.03	91.48
		6	**91.4**	**90.49**	**90.93**	**92.69**	91.82	**92.25**
	3e-5	1	84.36	83.74	83.98	86.98	85.84	86.15
		2	86.54	88.49	87.36	87.45	89.63	88.46
		3	88.06	89.17	88.53	90.76	90.04	90.33
		4	90.22	89.29	89.75	90.64	90.21	90.39
		5	90.77	90.34	90.51	90.96	91.05	90.97
		6	90.7	90.33	90.48	92.09	91.39	91.71
	2e-5	1	83.48	83.19	83.29	86.69	84.97	85.8
		2	87.68	86.92	87.2	87.38	89.43	88.29
		3	89.27	88.88	89.05	91.18	89.13	90.12
		4	90.86	87.81	89.26	90.76	89.82	90.26
		5	89.68	88.97	89.25	91.68	90.21	90.92
		6	90.23	89.56	89.86	91.23	90.18	90.69

Table B.2: Results of BERT and NetBERT on the validation set of the query classification dataset. The reported metrics are the *macro-average* Precision (P), Recall (R) and F1-score (F1), all expressed in percent. The best scores appear in **bold**, and the second best scores are underlined.

Appendix C

About Cisco Systems

Cisco Systems is an American multinational technology conglomerate headquartered in San Jose, California. Just as San Francisco – for which Cisco is named – provides a gateway to the Pacific Rim, Cisco provides the networking technology that is the gateway to computer-based communication. This Silicon Valley company is the worldwide market leader in routing, switching, unified communications, wireless communication, and security. It develops, manufactures and sells networking hardware, software, telecommunications equipment and other high-technology services which are used to create Internet solutions. Cisco has a firm belief that Internet will change the way people work, live, play and learn, and will also allow numerous leading enterprises and their partners to benefit from a "globally networked economy".

Cisco was founded in December 1984 by Leonard Bosack and Sandy Lerner, two computer scientists from Stanford University who were experimenting to connect detached networks in two separate buildings on Stanford campus. After running network cables between the two buildings, and connecting them with bridges and then routers, the two realized that to make the disparate networks talk to each other and share information, a new technology that could handle the different local area protocols was needed. Hence in 1986, Bosack and Lerner manufactured the world's first multi-protocol router, connecting different types of networks reliably and ushering in a communications revolution. By 1989, with only three products and 111 employees, Cisco's revenues were $27 million. With the start of widespread use of the Internet in the 1990s, Cisco earned its first patent for its method and equipment for routing communication among computer networks. In 1997, with thirty-three patents at hand, many leading-edge products and offices worldwide, the company introduced its first voice-over-IP and fax-over-IP products as well as a line of cable data products. The following year, Cisco introduced its first cable modem for small offices, homes, and telecommuting as well as the Gigabit Ethernet and Layer-3 routing in switches.

Today, Cisco continues to concentrate on its core areas of routing and switching as well as on advanced technologies such as IP communications, wireless LAN, home networks, network security, storage area networking, and video systems. As networking evolves from infrastructure to platform, Cisco again is at the center of a new way of communicating through secure convergence of data, voice, video, and mobile communication. Through an extensive R&D, Cisco is continuing to fulfill its promise to transform the way people connect, communicate, collaborate, and grow. Since its IPO in 1990, Cisco's revenue has grown from $69 million to $51.9 billion in 2019. The company now has nearly 76,000 employees around the world. It was ranked No.69 on the 2019 Fortune Global 500 list, and has been named the World's Most Admired Companies by the Fortune Magazine for many times.

Bibliography

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Edward O Wilson. *The meaning of human existence.* WW Norton & Company, 2014.

Andreas Hotho, Andreas Nürnberger, and Gerhard Paaß. A brief survey of text mining. In *Ldv Forum*, volume 20, pages 19–62. Citeseer, 2005.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper.pdf*, 2018.

Elizabeth D Liddy. Natural language processing. *Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc.*, 2001.

Claude E Shannon and Warren Weaver. The mathematical theory of information. *Urbana: University of Illinois Press*, 97, 1949.

Leon E Dostert. The georgetown-ibm experiment. *Machine translation of languages. John Wiley & Sons, New York*, pages 124–135, 1955.

Noam Chomsky. Syntactic structures. *The Hague: Mouton*, 1957.

John R Pierce, John B Carroll, Eric P Hamp, David G Hays, Charles F Hockett, Anthony G Oettinger, and Alan Perlis. Language and machines — computers in translation and linguistics. *ALPAC report, National Academy of Sciences, National Research Council, Washington, DC*, 1966.

Noam Chomsky. Aspects of the theory of syntax. *Cambridge: M.I.T. Press*, 1965.

Charles Fillmore. The case for case. *Bach and Harms (Ed.): Universals in Linguistic Theory*, 1968.

Allan M Collins, M Ross Quillian, et al. Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, 8(2):240 – 247, 1969.

William A Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, 1970.

Roger C Schank. Conceptual dependency: A theory of natural language understanding. *Cognitive psychology*, 3(4):552–631, 1972.

Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, Massachusetts Institute of Technology, Cambrdige Project, 1971.

W Woods, R Kaplan, and B Nash-Webber. The lunar sciences natural language information system: Final report (bolt, beranek and newman, cambridge, ma). *Woods discusses the LUNAR program which answers scientist's questions about the moon rocks*, 1972.

Kenneth Mark Colby. Ten criticisms of parry. *ACM SIGART Bulletin*, 48:5–9, 1974.

Roger C Schank and Robert P Abelson. Scripts, plans, and knowledge. In *IJCAI*, volume 75, pages 151–157, 1975.

James Richard Meehan. The metanovel: writing stories by computer. Technical report, Yale Univ. New Haven. Conn. Dept. of Computer Science, 1976.

Wendy G Lehnert. A conceptual theory of question answering. In *Proceedings of the 5th international joint conference on Artificial intelligence-Volume 1*, pages 158–164, 1977.

Richard Edward Cullingford. Script application: computer understanding of newspaper stories. Technical report, Yale Univ. New Haven. Conn. Dept. of Computer Science, 1978.

Roger C Schank and Robert Wilensky. A goal-directed production system for story understanding. In *Pattern-directed inference systems*, pages 415–430. Elsevier, 1978.

Jaime Guillermo Carbonell. Subjective understanding: computer models of belief systems. Technical report, Yale Univ. New Haven. Conn. Dept. of Computer Science, 1979.

Eugene Charniak. Passing markers: A theory of contextual influence in language comprehension. *Cognitive science*, 7(3):171–190, 1983.

Michael G Dyer. The role of affect in narratives. *Cognitive Science*, 7(3):211–242, 1983.

Christopher K Riesbeck and C Martin. Direct memory access parsing. *Experience, memory and reasoning*, pages 209–226, 1986.

Barbara J Grosz, Douglas E Appelt, Paul A Martin, and Fernando CN Pereira. Team: an experiment in the design of transportable natural-language interfaces. *Artificial Intelligence*, 32(2):173–243, 1987.

Graeme Hirst. Semantic interpretation and ambiguity. *Artificial Intelligence*, 34(2):131 – 177, 1987. ISSN 0004-3702.

Lalit R Bahl, Peter F Brown, Peter V de Souza, and Robert L Mercer. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(7):1001–1008, 1989.

Eric Brill, David Magerman, Mitchell Marcus, and Beatrice Santorini. Deducing linguistic structure from the statistics of large corpora. In *Proceedings of the 5th Jerusalem Conference on Information Technology, 1990.'Next Decade in Information Technology'*, pages 380–389. IEEE, 1990.

Mahesh V Chitrao and Ralph Grishman. Statistical parsing of messages. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*, 1990.

Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Frederick Jelinek, John Lafferty, Robert L Mercer, and Paul S Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1991.

Hideki Tanaka. Verbal case frame acquisition from a bilingual corpus: Gradual knowledge acquisition. In *Proceedings of the 15th conference on Computational linguistics-Volume 2*, pages 727–731. Association for Computational Linguistics, 1994.

Ilussein Allmuallim, Yasuhiro Akiba, Takefumi Yamazaki, Akio Yokoo, and Shigeo Kaneda. Two methods for learning translation rules from examples and a semantic hierarchy. In *COLING 1994 Volume 1: The 15th International Conference on Computational Linguistics*, 1994.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Michał Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. Frustratingly short attention spans in neural language modeling. *arXiv preprint arXiv:1702.04521*, 2017.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

David Mimno and Laure Thompson. The strange geometry of skip-gram with negative sampling. In *Empirical Methods in Natural Language Processing*, 2017.

Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Linear algebraic structure of word senses, with applications to polysemy. *Transactions of the Association for Computational Linguistics*, 6:483–495, 2018.

Maria Antoniak and David Mimno. Evaluating the stability of embedding-based word similarities. *Transactions of the Association for Computational Linguistics*, 6:107–119, 2018.

Laura Wendlandt, Jonathan K Kummerfeld, and Rada Mihalcea. Factors influencing the surprising instability of word embeddings. *arXiv preprint arXiv:1804.09692*, 2018.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Niki Parmar, Mike Schuster, Zhifeng Chen, et al. The best of both worlds: Combining recent advances in neural machine translation. *arXiv preprint arXiv:1804.09849*, 2018.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700, 2015.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in neural information processing systems*, pages 1693–1701, 2015.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087, 2015.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9, 2019.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764, 2019.

Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

Thomas K Landauer and Susan T Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211, 1997.

Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior research methods, instruments, & computers*, 28(2):203–208, 1996.

Douglas LT Rohde, Laura M Gonnerman, and David C Plaut. An improved method for deriving word meaning from lexical co-occurrence. *Cognitive Psychology*, 7:573–605, 2004.

Rémi Lebret and Ronan Collobert. Word emdeddings through hellinger pca. *arXiv preprint arXiv:1312.5542*, 2013.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5: 135–146, 2017.

Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

Oren Melamud, Jacob Goldberger, and Ido Dagan. context2vec: Learning generic context embedding with bidirectional lstm. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 51–61, 2016.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305, 2017.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. Ernie: Enhanced language representation with informative entities. *arXiv preprint arXiv:1905.07129*, 2019a.

Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. Clinicalbert: Modeling clinical notes and predicting hospital readmission. *arXiv preprint arXiv:1904.05342*, 2019.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, 2020.

Iz Beltagy, Arman Cohan, and Kyle Lo. Scibert: Pretrained contextualized embeddings for scientific text. *arXiv preprint arXiv:1903.10676*, 2019.

Jay Alammar. The Illustrated Transformer. Retrieved from http://jalammar.github.io/illustrated-transformer, 2018.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Jay Alammar. The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning). Retrieved from http://jalammar.github.io/illustrated-bert, 2019.

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

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019a.

Yu Sun, Shuohuan Wang, Yukun Li, Shikun Feng, Hao Tian, Hua Wu, and Haifeng Wang. Ernie 2.0: A continual pre-training framework for language understanding. *arXiv preprint arXiv:1907.12412*, 2019.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*, 2019.

Aiting Liu, Ziqi Huang, Hengtong Lu, Xiaojie Wang, and Caixia Yuan. Bb-kbqa: Bert-based knowledge base question answering. In *China National Conference on Chinese Computational Linguistics*, pages 81–92. Springer, 2019b.

Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*, 2019.

Nina Poerner, Ulli Waltinger, and Hinrich Schütze. Bert is not a knowledge base (yet): Factual knowledge vs. name-based reasoning in unsupervised qa. *arXiv preprint arXiv:1911.03681*, 2019.

Matthew E Peters, Mark Neumann, IV Logan, L Robert, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A Smith. Knowledge enhanced contextual word representations. *arXiv preprint arXiv:1909.04164*, 2019.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

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 preprint arXiv:1910.10683*, 2019.

C Rosset. Turing-nlg: A 17-billion-parameter language model by microsoft. *Microsoft Blog*, 2019.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher 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, 2020.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: The impact of student initialization on knowledge distillation. *arXiv preprint arXiv:1908.08962*, 2019.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Amir Bakarov. A survey of word embeddings evaluation methods. *arXiv preprint arXiv:1801.09536*, 2018.

Davide Chicco and Giuseppe Jurman. The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC genomics*, 21(1):6, 2020.

Bradley Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*, pages 569–593. Springer, 1992.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

Anand Rajaraman and Jeffrey David Ullman. Mining of massive datasets: Data mining (ch01). *Min. Massive Datasets*, 18:114–142, 2011.

Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013c.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*, 2019b.

Han Xiao. bert-as-service. https://github.com/hanxiao/bert-as-service, 2018.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Bin Wang and C-C Jay Kuo. Sbert-wk: A sentence embedding method by dissecting bert-based word models. *arXiv preprint arXiv:2002.06652*, 2020.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 2019.

Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28 (2):129–137, 1982.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2010.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*, 2017.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.