UNIVERSITY OF AMSTERDAM

MSc Artificial Intelligence

Master Thesis

Distilling BERT: Transfer Dataset Size and Composition Effects

by

Steven van de Graaf

12234036

July 1, 2021

48 ECTS
November 2019 - June 2021

Supervisor:
Dr. Ir. J. Kamps

Assessor:
Dr. I. Calixto

Institute for Logic,
Language and Computation

Abstract

As Transformer(-based) architectures in the field of *Natural Language Processing* (NLP) have gotten increasingly better, so have they gotten increasingly bigger. These *Neural Network* (NN) architectures often-times contain hundreds of millions of – and more recently hundreds of billions of – parameters, which hurts their sustainability and accessibility. It is no surprise then, that methods which aim at making these NNs more efficient have become increasingly popular. *Knowledge Distillation* (KD) is just one such method, in which a smaller, untrained student network is trained to mimic a larger, (pre-)trained teacher network. Research into KD as applied to Transformer(-based) architectures often-times use huge corpora, without any scientific foundation. This makes the KD process itself quite inefficient, ironically enough.

This thesis investigates the effects of the transfer dataset $\mathcal{D}_T$ used for the KD process, both in terms of *size* and in terms of *composition*. To this end, we distill the knowledge of $\text{BERT}_{\text{BASE}}$, our teacher network of choice, into our student network of choice, $\text{BERT}_{\text{STUDENT}}$, multiple times using transfer datasets of various sizes and compositions. After distillation we evaluate each distilled student network on the GLUE benchmark and on SQuAD.

Our investigation has yielded interesting results. When it comes to the *size* of transfer dataset $\mathcal{D}_T$, we show that it is possible to successfully distill the knowledge of the teacher into the student using a transfer dataset that is a fraction of the size of what is conventionally used. This all at marginal (if not negligible) cost in terms of downstream task performance. With respect to the *composition* of transfer dataset $\mathcal{D}_T$, we show that randomizing and/or generating sequences does come at a significant cost in terms of performance, in general. For some specific tasks, however, downstream performance remains competitive.

Acknowledgements

This Master Thesis is the ultimate deliverable that marks the end of a journey that officially started all the way back in November 2019. While the reader might have already done the math, this means it has been more than one year and a half since I first started working on this thesis (of what should have been only 9 months). To say that the process of writing this thesis proved to be a long, uphill battle would be an understatement.

During this trying time I received a great deal of support, however. As such, my sincere thanks and acknowledgements of their support and patience is in order.

I would first and foremost like to thank my supervisor, dr. ir. Jaap Kamps. Thank you first and foremost for your seemingly inexhaustible patience. I would also like to thank him for the many stimulating conversations, for this academic expertise and for his keen insights. I am grateful for your support and guidance throughout this process.

Next, I would like to thank my family. Thank you Astrid for your love and support. I could not have finished this thesis without either. A big thanks is also due to Hans and Anita. I am grateful for the care, patience and accountability you have provided me during this period of my life. Lastly, I want to thank Jamey and Eline for being kind enough to let me make use of their space in their absence.

Finally, I would like to thank all my friends that have helped me along the way, whether through reassuring conversations, joyful distractions or otherwise. Special thanks are in order for Han, Azamat and Nikos, who have each proof-read parts of this thesis.

Contents

List of Abbreviations

AT Attention Transfer.

BERT Bidirectional Encoder Representations from Transformers.
BiLSTM Bidirectional LSTM.
BiRNN Bidirectional RNN.
BoT Bag-of-Tokens.
BoW Bag-of-Words.

CDF cumulative distribution function.
CE Cross Entropy.
CoLA Corpus of Linguistic Acceptability.
CV Computer Vision.

EM Exact Match.

FFN Feed-Forward Network.
FLOPS Floating Point Operations Per Second.
FMT Feature Map Transfer.
FP32 32-bit Floating Point.

GLUE General Language Understanding Evaluation.
GPT Generative Pre-Training.

KD Knowledge Distillation.
KL Kullback-Leibler divergence.

LM Language Model.
LSTM Long Short-Term Memory.

MLM Masked Language Model.

MNLI Multi-Genre NLI.

MoE Mixture of Experts.

MRPC Microsoft Research Paraphrase Corpus.

MSE Mean Squared Error.

MT-DNN Multi-Task Deep Neural Network.

MTL Multi-Task Learning.

NL Natural Language.

NLG Natural Language Generation.

NLI Natural Language Inference.

NLP Natural Language Processing.

NLU Natural Language Understanding.

NMT Neural Machine Translation.

NN Neural Network.

NSP Next Sentence Prediction.

PD Pre-training Distillation.

PD Pre-trained Distillation.

PKD Patient Knowledge Distillation.

QA Question Answering.

QNLI Question-answering NLI.

QQP Quora Question Pairs.

ReLU Rectified Linear Unit.

RNN Recurrent Neural Network.

RoBERTa Robustly optimized BERT approach.

RTE Recognizing Textual Entailment.

seq2seq sequence-to-sequence.

SOTA State-Of-The-Art.

SQuAD Stanford Question Answering Dataset.

SST Stanford Sentiment Treebank.

STL Single-Task Learning.

STS-B Semantic Textual Similarity Benchmark.

TBC Toronto BookCorpus.

WNLI Winograd NLI.

List of Figures

List of Tables

1 Introduction

This chapter (unsurprisingly) serves to introduce the reader to the subject of this Master Thesis. As such, it attempts to answer important "Wh"-questions, primarily questions pertaining to "what" and "why".

In an attempt to answer the "why" question, this chapter starts off with the motivation for this thesis in Section 1.1. To this end, the reader is presented a big-picture overview of some current trends in the field of NLP, and the various (negative) impacts these trends have. Moreover, a promising technique to mediate these effects is introduced, along with some important gaps in knowledge concerning this technique, which will be expanded upon in Chapter 2.

Next, these knowledge gaps are addressed in the form of a set of research questions in Section 1.2. These research questions steer the direction of the current research and should provide a rather concrete answer to the "what" question.

Finally, the outline of this thesis is given in Section 1.3.

1.1 Motivation

For roughly three years now, the field of NLP has seen rapid, significant advances in the domains of *Natural Language Understanding* (NLU) and *Natural Language Generation* (NLG). These advances have led some to claim that NLP is having its "ImageNet moment" (Ruder, 2018), referring to the big boom experienced in the field of *Computer Vision* (CV) after 2012, when AlexNet (Krizhevsky et al., 2012) made significant advances on the ImageNet challenge (Russakovsky et al., 2015).

This "big boom" is exemplified by the frequency at which the *State-Of-The-Art* (SOTA) results on various benchmarks and downstream performance tasks like GLUE[1] (Wang et al., 2018) and SQuAD[2] (Rajpurkar et al., 2016, 2018) are achieved, with new leaders on the leaderboard nearly every month or two.

These advances are largely thanks due to so-called "Transformer"-based architectures (Vaswani et al., 2017), which dispense with re-

[1] The GLUE leaderboard can be found here: https://gluebenchmark.com/leaderboard

[2] The SQuAD leaderboard can be found here: https://rajpurkar.github.io/SQuAD-explorer/

currence and convolutions in favor of attention mechanisms. While proven effective, these architectures often-times contain hundreds of millions of – and more recently trillions of – parameters, as can be seen in Table 1.1. As such, they do come at a significant cost in terms of sustainability (both financially and environmentally) and accessibility, both of which we will further expand upon in this section.

Paper	Date	Name	Number of Parameters
Radford et al. (2018)	06-2018	GPT	110 000 000
Devlin et al. (2018)	10-2018	$\text{BERT}_{\text{LARGE}}$	320 000 000
Radford et al. (2019)	02-2019	GPT-2	1 542 000 000
Shoeybi et al. (2019)	09-2019	Megatron	8 300 000 000
Raffel et al. (2019)	10-2019	T5-11B	11 000 000 000
Microsoft (2020)	02-2020	Turing-NLG	17 000 000 000
Brown et al. (2020)	05-2020	GPT-3	175 000 000 000
Fedus et al. (2021)	01-2021	Switch-C	1 571 000 000 000[3]

Table 1.1: Sizes of recent Transformer(-based) architectures in terms of the number of parameters.

The (lack of) sustainability of these NNs is exemplified both by the financial costs and the CO_2 emissions involved in the process of training these networks. As reported by Devlin et al. (2018), their $\text{BERT}_{\text{BASE}}$ network was trained on 4 Cloud TPU v2 Pods (containing 16 Cloud TPU v2 chips in total) for 4 days (96 hours), which would cost somewhere between \$4.1 k - \$5.7 k[4] and produce about 650 kg of CO_2 emissions[5] to train. (Lacoste et al., 2019; Strubell et al., 2019) More recently, Brown et al. (2020) report training their GPT-3 network on NVIDIA Tesla V100 GPUs, which required 3.14×10^{23} *Floating Point Operations Per Second* (FLOPS). Even with the most optimistic estimation, this would require 304 GPU-*years*, cost roughly \$3.0 M[6] and produce about 2.952×10^5 kg of CO_2 emissions.

These figures do not even yet include an estimate of the costs involved during the hyper-parameter search, however, nor do they include an estimate of the costs involved during inference (when deployed in a production environment, for example), both of which are expected to be significantly larger (in the long run) than the costs involved in training. As such, the sustainability of these Transformer-based architectures is highly questionable.

Despite recent effort to make these these NNs more accessible[7], their inherent size make it difficult for anyone without (access to) very expensive hardware to make use of them. While using a huge NN like GPT-3 (which contains 175 B parameters) would be unattainable for anyone without the financial backing of a big technology company, even using a "smaller" architecture like $\text{BERT}_{\text{BASE}}$ (which contains "only" 110 M parameters) isn't easy, nor cheap. This not only hurts the democratization of these technologies, but also their reproducibility, which is a core principal of the scientific method.

This all highlights the increasing importance of methods which aim at making these NNs more efficient, exemplified by the recent advent of "Green AI" (Schwartz et al., 2019). One such method is KD (Buciluǎ et al., 2006; Hinton et al., 2015), which can best be described

[3] The authors leverage the *Mixture of Experts* (MoE) paradigm to create a NN architecture that is sparsely-activated. As such, it contains an outrageous number of parameters, but the computational cost is much smaller and constant.

[4] The most recent prices for TPU pods can be found here: https://cloud.google.com/tpu/pricing

[5] CO_2 emissions were estimated using the MachineLearning Impact calculator.

[6] The most recent prices for GPUs in Google Cloud can be found here: https://cloud.google.com/compute/gpus-pricing

[7] Examples include the Hugging Face's "Model Hub"

as a teacher-student framework for knowledge transfer in which a smaller, untrained student network is trained to mimic a larger, pre-trained teacher network. As such, the larger (pre-trained) teacher network is effectively "compressed" into the smaller (untrained) student network. This allows for comparable performance, while requiring less computational resources.

In the last two years or so, there have been various efforts to apply KD to Transformer-based architectures (Sun et al., 2019; Jiao et al., 2019; Sanh et al., 2019; Turc et al., 2019), resulting in NNs which are both smaller and faster, while still achieving comparable performance. While promising, this new line of research is still immature, as reflected by the lack of established best-practices when it comes to the various aspects of KD, such as: initialization strategy of the student network, composition of the loss function, the amount of data required, the type of data required, the (relative) importance of fine-tuning over pre-training, among others.

1.2 *Research questions*

This lack of established best-practices for KD as applied to Transformer-based architectures is precisely what this research aims to address. This is still too big a scope for this thesis, however, as there are too many variables to possibly consider. As such, instead of investigating the effects of all possible aspects of KD as applied to a variety of Transformer-based architectures, we will first select a specific Transformer-based architecture for the purposes of this thesis, followed by the selection of a more narrow set of aspects of KD to consider.

While the biggest NNs (containing hundreds of billions of parameters) like GPT-3 by Brown et al. (2020) stand to gain the most from being compressed, this research focuses its efforts solely on the popular $\text{BERT}_{\text{BASE}}$ network by Devlin et al. (2018). The motivation for this choice is three-fold. First, as $\text{BERT}_{\text{BASE}}$ contains "only" 110 M parameters, it is well within the limits of our (computational) resources to use it in our research. Second, previous research into KD as applied to Transformer-based architectures (see Chapter 3) also focus their efforts (almost) exclusively on $\text{BERT}_{\text{BASE}}$ (and/or $\text{BERT}_{\text{LARGE}}$, its "bigger brother"). Third and last, the $\text{BERT}_{\text{BASE}}$ architecture is a good representation of Transformer-based architectures in general, as it is almost identical to the encoder stack of the original "Transformer" architecture by Vaswani et al. (2017), differing only in size.

When it comes to what aspects of the KD process to investigate, this research has settled on the data used during the pre-training stage (i.e. the transfer dataset $\mathcal{D}_T$). While KD is a method to compress NNs, thereby making them more efficient, the KD process as employed in previous research itself can be quite inefficient, ironically

enough. This is primarily due to the usage of large text corpora in the pre-training stage, which require significant compute. As such, our first research question relates to the size of the transfer dataset $\mathcal{D}_T$ used during pre-training in the KD process as follows:

RQ$_1$ What are the effects of the *size* of the transfer dataset $\mathcal{D}_T$ used during pre-training in the Knowledge Distillation process as applied to BERT$_{\text{BASE}}$, measured in performance on downstream performance task(s)?

Another drawback of using large text corpora when it comes to distilling Transformer-based architectures is that corpora that are both high in quality and quantity are scarce. What's more, even when they are available, they often-times require significant pre-processing before they can be used in a KD-setting. It is not a strict requirement that proper *Natural Language* (NL) data be used in the KD process, however. Instead, it might be possible to use synthetic data to transfer the knowledge of the teacher network to the student network. As such, our second research question relates to the composition of the transfer dataset $\mathcal{D}_T$ used during pre-training in the KD process as follows:

RQ$_2$ What are the effects of the *composition* of the transfer dataset $\mathcal{D}_T$ used during pre-training in the Knowledge Distillation as applied to BERT$_{\text{BASE}}$, measured in performance on downstream performance task(s)?

1.3 Outline

In the following chapter (Chapter 2), the background concepts that this thesis builds upon are presented to the reader. Among others, these background concepts include the Transformer architecture in general and BERT specifically, the neural network compression method known as Knowledge Distillation, and the downstream performance tasks used to evaluate our approach.

Chapter 3 presents a comprehensive overview of work related to the present research. In other words: research that has to do with Knowledge Distillation applied to Transformer(-based) architectures.

In Chapter 4, we review our experimental design. This includes our student and teacher NN architectures of choice, our pre-training procedure, our evaluation procedure and the baselines to which our results will be compared. Also included in this chapter is a note on compute power, an exhaustive overview of hyperparameters (for both pre-training and evaluation), and a comparison of datasets.

Our experiments and their results are introduced in Chapter 5. Our first experiment attempts to answer **RQ**$_1$, and, as such, investigates the relationship between the *size* of the transfer dataset $\mathcal{D}_T$ and downstream task performance. Conversely, our second experiment attempts to answer **RQ**$_2$, and, investigates the effects of the *composition* of the transfer dataset $\mathcal{D}_T$ on downstream task performance.

Lastly, our conclusions are presented in Chapter 6.

2 Background

Having introduced the subject of this Master Thesis in the previous chapter (by means of answers to important "Wh"-questions), this chapter dives deeper into the background concepts that this research builds upon. Specifically, this chapter aims to answer the following questions:

- What is the Transformer architecture in general and BERT specifically?
- What is Knowledge Distillation?
- How do we evaluate our approach?

To this end, this chapter first introduces the *sequence-to-sequence* (seq2seq) architecture in Section 2.1, which has brought about significant advances in transduction tasks, such as translation, question answering and speech recognition, among others. The original or "conventional" seq2seq architecture relies on recurrence to compute representations of its input and output, which does come with certain drawbacks, however. As such, an important improvement of the conventional seq2seq architecture is introduced next, which relies on attention mechanisms.

Next, the inner workings of the *Transformer* architecture are detailed in Section 2.2. As a deeper, attention-only seq2seq architecture, it has been responsible for a recent wave of advancements in the domains of *Natural Language Understanding* (NLU) and *Natural Language Generation* (NLG).

Third, the $\text{BERT}_{\text{BASE}}$ architecture is reviewed in Section 2.3, highlighting the differences between it and the original Transformer architecture. As mentioned previously in Chapter 1, this is the particular Transformer-based architecture used in the current research.

Penultimately, Section 2.4 introduces *Knowledge Distillation* (KD), a *Neural Network* (NN) compression method that has gained popularity in recent years, especially when it comes to Transformer(-based) architectures (see Chapter 3). Again, as introduced in the previous chapter, this is the particular NN compression method under investigation in this research.

Finally, Section 2.5 presents the methods with which our approach will be evaluated in the present work. Specifically, we'll first detail the *General Language Understanding Evaluation* (GLUE) benchmark, a

collection of 9 NLU tasks (with accompanying datasets), before introducing the *Stanford Question Answering Dataset* (SQuAD), a sizable reading comprehension dataset.

2.1 Sequence-to-sequence Architectures

2.1.1 Conventional sequence-to-sequence architectures

Originally developed for the purposes of *Neural Machine Translation* (NMT), conventional *sequence-to-sequence* (seq2seq) NN architectures (Sutskever et al., 2014; Cho et al., 2014b) rely on *Recurrent Neural Networks* (RNNs) to compute representations of their input and output. These NNs are composed of an *encoder*, which encodes an input (or "source") sequence into a fixed-length vector (often called a "context" or "thought" vector), and a *decoder*, which uses this encoded vector to compute the output (or "target") sequence, as illustrated in Fig. 2.1.

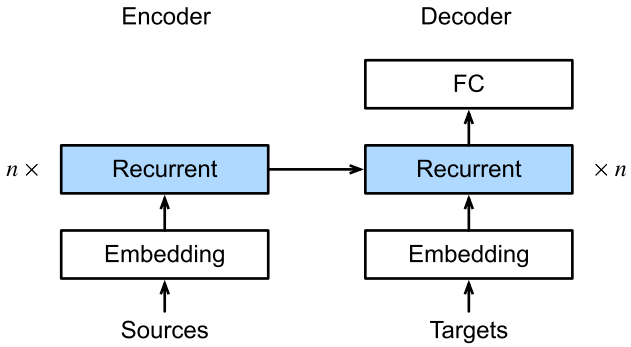

Figure 2.1: The sequence-to-sequence network architecture. Source: Dive into Deep Learning

Formally, given an input sequence $\boldsymbol{x} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_N \end{bmatrix}^\top$ of length N, the encoder of conventional seq2seq networks encodes this input sequence $\boldsymbol{x}$ into a context vector $\boldsymbol{c}$ of length M as follows:

$$\mathbf{h}_i = f(\boldsymbol{x}_i, \boldsymbol{h}_{i-1}), \tag{2.1}$$

$$\mathbf{c} = q(\begin{Bmatrix} \mathbf{h}_1, & \mathbf{h}_2, & \cdots, & \mathbf{h}_N \end{Bmatrix}), \tag{2.2}$$

where $\boldsymbol{h}_n \in \mathbb{R}^M$ is the hidden state of the encoder at step t and f and q are some non-linear functions. For example, Sutskever et al. (2014) use

$$\begin{aligned} \mathbf{h}_i &= f(\boldsymbol{x}_i, \boldsymbol{h}_{i-1}) = \text{LSTM}(\boldsymbol{x}_i, \boldsymbol{h}_{i-1}), \\ \mathbf{c} &= q(\begin{Bmatrix} \mathbf{h}_1, & \mathbf{h}_2, & \cdots, & \mathbf{h}_N \end{Bmatrix}) = \boldsymbol{h}_N, \end{aligned}$$

where LSTM is the *Long Short-Term Memory* (Hochreiter and Schmidhuber, 1997).

The decoder now uses this context vector $\boldsymbol{c}$ and any previously predicted elements of the output sequence $\begin{Bmatrix} y_1, & y_2, & \cdots, & y_{t-1} \end{Bmatrix}$ to

predict the next element of the output sequence y_t. This joint probability is given by:

$$p(\boldsymbol{y}) = \prod_{t=1}^{T} p\left(y_t | \boldsymbol{c}, \left\{y_1, \quad y_2, \quad \cdots, \quad y_{t-1}\right\}\right), \tag{2.3}$$

where $\boldsymbol{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_T \end{bmatrix}^\top$ is the output sequence of length T. Again using an RNN for the decoder, the hidden state $\boldsymbol{s}_t$ of the decoder is given by:

$$\boldsymbol{s}_t = f(\boldsymbol{c}, y_{t-1}, \boldsymbol{s}_{t-1}), \tag{2.4}$$

$$\mathbf{s}_{t=0} = \boldsymbol{h}_N, \tag{2.5}$$

where f is the same non-linear function as used for the encoder, and the hidden state of the encoder at the last step $\boldsymbol{h}_N$ is used as the initial hidden state of the decoder $\boldsymbol{s}_{t=0}$. As such, we can compute the conditional probabilities in Eq. (2.3) as follows:

$$p\left(y_t | \boldsymbol{c}, \left\{y_1, \quad y_2, \quad \cdots, \quad y_{t-1}\right\}\right) = g(\boldsymbol{c}, y_{t-1}, \boldsymbol{s}_t), \tag{2.6}$$

where g is again some non-linear function.

2.1.2 *Sequence-to-sequence architectures with attention*

While these conventional seq2seq models perform well for relatively short sequences, they do suffer from deteriorating performance as the length of the sequence increases (Cho et al., 2014a). To address this issue, attention mechanisms were introduced as an extension to the conventional encoder-decoder architecture (Bahdanau et al., 2014; Luong et al., 2015), as illustrated in Fig. 2.2, which significantly outperform their conventional counterparts, regardless of the sequence length.

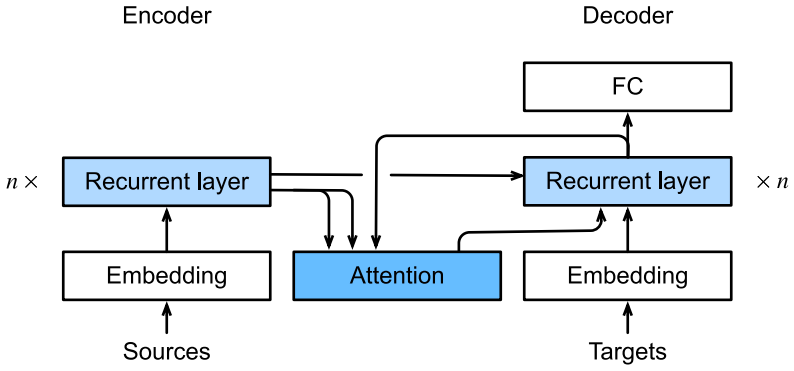

Figure 2.2: The sequence-to-sequence network architecture with attention mechanisms. Source: Dive into Deep Learning

Instead of using a single fixed-length context vector $\boldsymbol{c}$ as an encoding for the entire input sequence, these seq2seq models with attention mechanism(s) use a (variable length) sequence of context vectors for the encoding instead. Whenever the decoder generates a new element of the output sequence, it soft-searches for the most informative elements in the input sequence, and then uses the context vectors

associated with these most informative elements and all previously generated output elements as input.

As such, Eqs. (2.4) and (2.6) can be updated as follows:

$$\mathbf{s}_t = f(\boldsymbol{c}_t, y_{t-1}, \boldsymbol{s}_{t-1}), \tag{2.7}$$

$$p\left(y_t | \boldsymbol{x}, \left\{y_1, \quad y_2, \quad \cdots, \quad y_{t-1}\right\}\right) = g(\boldsymbol{c}_t, y_{t-1}, \boldsymbol{s}_t), \tag{2.8}$$

where for each element of the output sequence y_t, $\boldsymbol{c}_t$ is a distinct context vector which depends on all hidden states of the encoder $\left\{\mathbf{h}_1, \quad \mathbf{h}_2, \quad \cdots, \quad \mathbf{h}_N\right\}$ as follows:

$$\boldsymbol{c}_t = \sum_{i=1}^{N} \alpha_{t,i} \boldsymbol{h}_i, \tag{2.9}$$

where $\alpha_{t,i}$ serves as a weight of each encoder hidden state $\boldsymbol{h}_i$

$$\alpha_{t,i} = \frac{\exp\left(\text{score}\left(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i\right)\right)}{\sum_{i'=1}^{N} \exp\left(\text{score}\left(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}\right)\right)}. \tag{2.10}$$

In fact, $\alpha_{t,:}$ describes a probability distribution over the N encoder hidden states $\left\{\mathbf{h}_1, \quad \mathbf{h}_2, \quad \cdots, \quad \mathbf{h}_N\right\}$ using the softmax function over the *alignment score* between decoder hidden state $\boldsymbol{s}_{t-1}$ and encoder hidden state $\boldsymbol{h}_i$. Here, Bahdanau et al. (2014) parameterize this alignment score function as a simple *Feed-Forward Network* (FFN), which is trained jointly with the encoder and decoder:

$$\text{score}\left(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i\right) = \boldsymbol{V}_\alpha^\top \tanh\left(\boldsymbol{W}_\alpha \left[\boldsymbol{s}_{t-1}; \boldsymbol{h}_i\right]\right), \tag{2.11}$$

where $\boldsymbol{V}_\alpha \in \mathbb{R}^M$, $\boldsymbol{W}_\alpha \in \mathbb{R}^{M \times 2M}$ are learnable weight matrices.[1]

[1] Actually, Bahdanau et al. (2014) use a *Bidirectional RNN* (BiRNN) for the encoder, such that every encoder hidden state $\boldsymbol{h}_i$ is a concatenation of the hidden state $\overrightarrow{\boldsymbol{h}_i}$ of the forward RNN $\overrightarrow{f}$ and the hidden state $\overleftarrow{\boldsymbol{h}_i}$ of the backward RNN $\overleftarrow{f}$:

$$\mathbf{h}_i = \left[\overrightarrow{\boldsymbol{h}_i}; \overleftarrow{\boldsymbol{h}_i}\right].$$

As such, the alignment score in Eq. (2.11) should actually be:

$$\text{score}\left(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i\right) = \boldsymbol{V}_\alpha^\top \tanh\left(\boldsymbol{W}_\alpha \boldsymbol{s}_{t-1} + \boldsymbol{U}_\alpha \boldsymbol{h}_i\right),$$

where $\boldsymbol{V}_\alpha \in \mathbb{R}^M$, $\boldsymbol{W}_\alpha \in \mathbb{R}^{M \times M}$ and $\boldsymbol{U}_\alpha \in \mathbb{R}^{M \times 2M}$ are learnable weight matrices.

2.2 *Transformer Architecture*

Even though seq2seq architectures with attention mechanisms outperform their conventional counterparts (especially for longer sequences), they still rely on recurrence to compute representations of their input and output. This makes them inherently sequential in nature, as the hidden state $\boldsymbol{h}_i$ is still a function of the input for position i and the previous hidden state $\boldsymbol{h}_{i-1}$. Their sequential nature disallows for parallel computation of these hidden states, making their run time super-linear in the length of the input and target sequence (Kalchbrenner et al., 2016; Gehring et al., 2017).

The *Transformer* architecture (Vaswani et al., 2017) addresses this drawback by completely dispensing with recurrence altogether, instead

relying entirely on (self-)attention mechanisms to compute representations of the input and output sequences. It does so by using encoder and decoder stacks, which are both comprised of stacked self-attention and position-wise fully connected layers, as illustrated in Fig. 2.3. In the following subsections, all components of the Transformer architecture will be explained in detail.

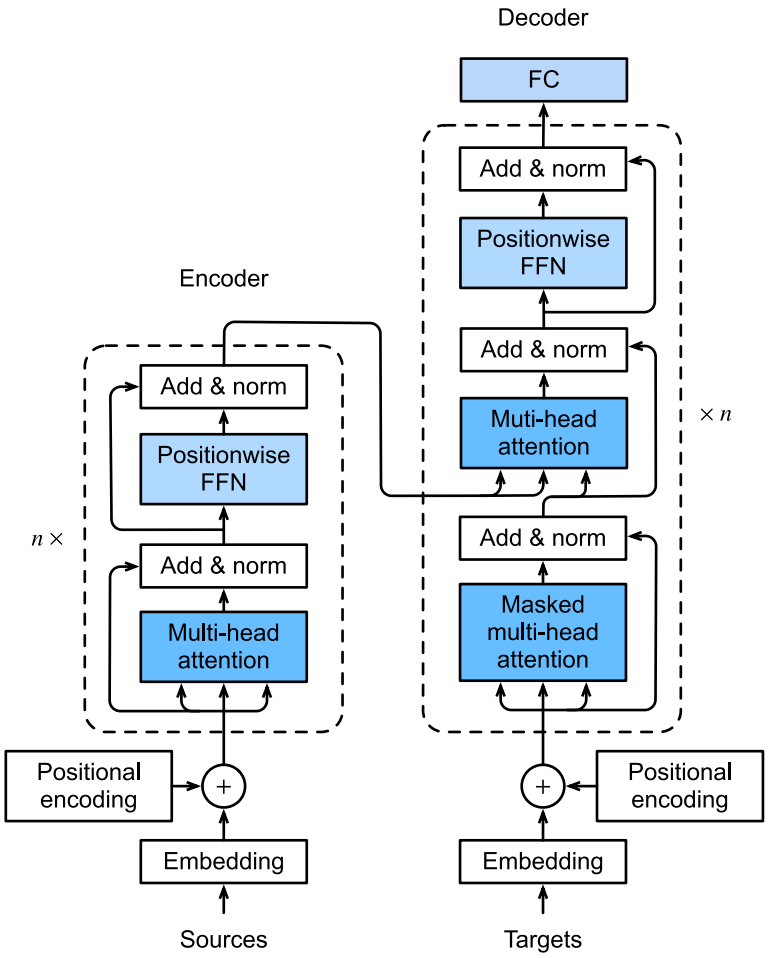

Figure 2.3: The Transformer network architecture. Source: Dive into Deep Learning

2.2.1 *Encoder and Decoder*

Encoder

The encoder stack consists of $N = 6$ (identical) layers, with two sub-layers each. The first sub-layer is a multi-head attention layer, and the second is a position-wise FFN. Both sub-layers have residual connections around them, followed by layer normalization. All outputs of the embedding layers and sub-layers in the model are of dimension $d_{\text{model}} = 512$.

Decoder

Similarly, the decoder stack also consists of $N = 6$ (identical) layers, although now with three sub-layers each. The first sub-layer is a masked multi-head attention layer, such that positions in the output

sequence are prevented from attending to subsequent positions. The second sub-layer is a multi-head attention layer over the output of the encoder stack. Finally, the third sub-layer is again a position-wise FFN. As with the encoder, all sub-layers feature residual connections, followed by layer normalization.

2.2.2 *Attention*

Scaled Dot-Product Attention

Vaswani et al. (2017) introduce a new form of attention called *scaled dot-product attention* that is highly similar to the *dot-product attention* of Luong et al. (2015). In their definition, they use the $(query, key, value)$ triplet representation, where the keys and values correspond with the encoder hidden state $\boldsymbol{h}_i$ and the query corresponds with the decoder hidden state $\boldsymbol{s}_{t-1}$. Now, formally, the alignment scoring function is defined as a scaled dot-product:

$$\text{score}\left(\boldsymbol{q}, \boldsymbol{k}\right) = \frac{\boldsymbol{q}^\top \boldsymbol{k}}{\sqrt{d_k}}, \tag{2.12}$$

where d_k is the dimension of the keys. In terms of the encoder hidden state $\boldsymbol{h}_i$ and the decoder hidden state $\boldsymbol{s}_{t-1}$, we get:

$$\text{score}\left(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i\right) = \frac{\boldsymbol{s}_{t-1}^\top \boldsymbol{h}_i}{\sqrt{N}}, \tag{2.13}$$

where N is the number of encoder hidden states $\left\{\mathbf{h}_1, \quad \mathbf{h}_2, \quad \cdots, \quad \mathbf{h}_N\right\}$. This alternative scoring function can then be plugged into Eq. (2.10) to compute the context vector $\boldsymbol{c}_i$ using Eq. (2.9).

In practice, however, all queries are packed together into a matrix $\boldsymbol{Q}$. Similarly, the keys and values are also packed together into matrices $\boldsymbol{K}$ and $\boldsymbol{V}$, respectively. The matrix of context vectors $\boldsymbol{C}$ is then given by:

$$\mathbf{C} = \text{Attention}\left(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\right) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}. \tag{2.14}$$

Similarly, stacking the encoder hidden states $\left\{\mathbf{h}_1, \quad \mathbf{h}_2, \quad \cdots, \quad \mathbf{h}_N\right\}$ into a matrix $\boldsymbol{H}$ and the decoder hidden states $\left\{\mathbf{s}_1, \quad \mathbf{s}_2, \quad \cdots, \quad \mathbf{s}_{t-1}\right\}$ into a matrix $\boldsymbol{S}$, we get:

$$\mathbf{C} = \text{softmax}\left(\frac{\boldsymbol{S}\boldsymbol{H}^\top}{\sqrt{N}}\right)\boldsymbol{H}. \tag{2.15}$$

Multi-Head Attention

Vaswani et al. (2017) found it beneficial to linearly project the queries, keys and values h times using different, learned linear projections to d_k, d_k, and d_v, respectively, instead of performing a single attention

function with d_{model}-dimensional queries, keys and values. The attention function is then applied in parallel on each of these projections, which yield d_v-dimensional output values. Finally, these are concatenated and projected once again in order to come to the final values, as illustrated in Fig. 2.4.

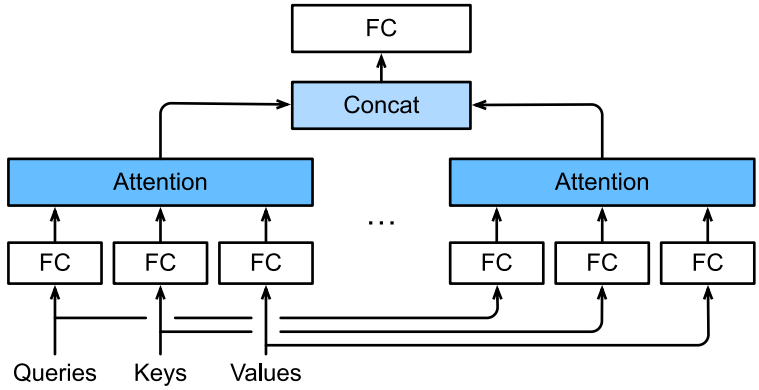

Figure 2.4: Multi-Head Attention. Source: Dive into Deep Learning

Formally, we have:

$$\text{MultiHead}\left(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}\right) = \left[\text{head}_1; \quad \text{head}_2; \quad \cdots \quad \text{head}_h\right] \boldsymbol{W}^O, \quad (2.16)$$

where

$$\text{head}_i = \text{Attention}\left(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V\right), \quad (2.17)$$

and where

$$\begin{aligned} \mathbf{W}_i^Q &\in \mathbb{R}^{d_{\text{model}} \times d_k}, \\ \mathbf{W}_i^K &\in \mathbb{R}^{d_{\text{model}} \times d_k}, \\ \mathbf{W}_i^V &\in \mathbb{R}^{d_{\text{model}} \times d_v}, \\ \text{and } \boldsymbol{W}^O &\in \mathbb{R}^{hd_v \times d_{\text{model}}} \end{aligned}$$

are the projection matrices. Vaswani et al. (2017) use $h = 8$ attention heads with $d_k = d_v = d_{\text{model}}/h = 64$.

2.2.3 *Position-wise Feed-Forward Networks*

The position-wise FFN present in each encoder and decoder layer has only a single hidden layer, coupled with a *Rectified Linear Unit* (ReLU) activation:

$$\text{FFN}(\boldsymbol{x}) = \max\left(0, \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1\right)\boldsymbol{W}_2 + \boldsymbol{b}_2, \quad (2.18)$$

where

$$\begin{aligned}\mathbf{W}_1 &\in \mathbb{R}^{d_{\text{model}} \times d_{ff}},\\ \mathbf{b}_1 &\in \mathbb{R}^{d_{ff}},\\ \mathbf{W}_2 &\in \mathbb{R}^{d_{ff} \times d_{\text{model}}},\\ \text{and } \boldsymbol{b}_2 &\in \mathbb{R}^{d_{\text{model}}}.\end{aligned}$$

The linear transformations use different parameters for each layer, but are applied the same across different positions. Vaswani et al. (2017) use $d_{ff} = 2048$.

2.2.4 *Embeddings*

For both the input and output sequences, Vaswani et al. (2017) use learned embeddings to convert their elements into vectors of dimension d_{model}, such that a sequence vector $\boldsymbol{z} = \begin{bmatrix} z_1 & z_2 & \cdots & z_l \end{bmatrix} \in \mathbb{R}^l$ becomes a sequence embedding matrix $\boldsymbol{Z} \in \mathbb{R}^{l \times d_{\text{model}}}$.

2.2.5 *Positional Encoding*

In order to preserve information about the relative or absolute position of elements in the input and output sequences, Vaswani et al. (2017) add fixed (i.e. not learned) *positional encodings* to the embeddings of the sequences, as illustrated in Fig. 2.5. The positional encodings have the same dimension d_{model} as the embeddings, such that the two can be summed.

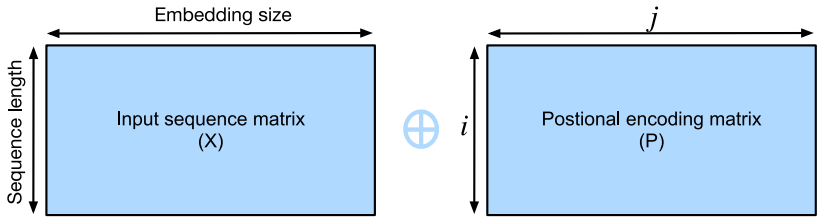

Figure 2.5: Positional encoding. Source: Dive into Deep Learning

For a sequence embedding $\boldsymbol{Z} \in \mathbb{R}^{l \times d_{\text{model}}}$, the positional encoding matrix $PE \in \mathbb{R}^{l \times d_{\text{model}}}$ is given by:

$$PE_{i,2j} = \sin\left(i/10000^{2j/d_{\text{model}}}\right) \tag{2.19}$$

$$PE_{i,2j+1} = \cos\left(i/10000^{2j/d_{\text{model}}}\right), \tag{2.20}$$

for $i = 1, 2, \ldots, l$ and $j = 1, 2, \ldots, \lfloor d_{\text{model}}/2 \rfloor$, where $\lfloor \cdot \rfloor$ denotes rounding down to the nearest integer.

While Vaswani et al. (2017) use fixed positional encodings in their work, it is also possible to use learned positional encodings instead (Devlin et al., 2018).

2.3 BERT

A little over two years ago, Devlin et al. (2018) introduced their new language representation NN *Bidirectional Encoder Representations from Transformers* (BERT).

BERT is a Transformer-based architecture that uses (only) the *Encoder* stack of the Transformer architecture (Vaswani et al., 2017), unlike OpenAI's *Generative Pre-Training* (GPT) and GPT-2 networks (Radford et al., 2018, 2019), which use (only) the *Decoder* stack instead, for example. The latter networks use a *Language Model* (LM) objective during pre-training, which involves predicting the next token in a sequence of tokens. This objective requires that these networks are unidirectional, however, such that when computing the self-attention for every token in the sequence, it can only attend to previous tokens. Otherwise the NN would learn that the prediction target is the next token in the sequence, which it would then always correctly predict, instead of learning any syntax, grammar or semantics. This unidirectionality requirement significantly limits the choice of architecture.

2.3.1 Pre-training Objectives

Masked Language Model

To address the unidirectionality constraint, Devlin et al. (2018) use a *Masked Language Model* (MLM) objective during pre-training instead. With this objective, some of the tokens in the sequence are randomly masked out and the goal is to predict the original tokens, based only on the context. This objective allows for using both the left and right context, thereby enabling the pre-training of a deep bidirectional Transformer-based NN.

Specifically, 15 % of the tokens in each sequence are randomly chosen to need a prediction. Of these tokens, 80 % will be actually masked out with the `[MASK]` token, 10 % will be replaced with a random token and 10 % will remain unchanged. The reasoning by Devlin et al. (2018) for not always replacing the "masked" word with the `[MASK]` token is that the `[MASK]` does not appear during the fine-tuning stage, resulting in a mismatch between pre-training and fine-tuning.

Next Sentence Prediction

In addition to the MLM objective, a *Next Sentence Prediction* (NSP) objective is used to jointly pre-train text-pair representations, which is useful for many down-stream tasks that are based on the relationship between two sentences, like *Question Answering* (QA) and *Natural Language Inference* (NLI), for example. Specifically, for each pre-training example, 50 % of the time, sentence `B` is the actual next sentence that follows sentence `A`, which is labeled as `IsNext`. The other 50 % of the time, sentence `B` is randomly chosen sentence from

the training corpus, which is labeled as `NotNext`.

Data

For their pre-training corpus, Devlin et al. (2018) use a concatenation of the *Toronto BookCorpus* (TBC) and English Wikipedia (see Section 4.3.2). This results in a corpus of roughly 3.3 B words in total.

2.3.2 *Input Representations*

BERT uses an input representation that is able to represent both single sentences and sentence pairs, both of which are called a *sequence*. To this end, WordPiece token embeddings (Wu et al., 2016) with a vocabulary size of 30522 tokens are used to encode every word in the sequence. The first token of every sequence is always the classification token `[CLS]`.

Furthermore, sentence pairs are packed together into a single sequence, which are differentiated both by the `[SEP]` token in between the two sentences, but also by the use of segment embeddings, indicating whether a token belongs to sentence `A` or sentence `B`.

Finally, the representation of any input sequence is given by the sum of the token embeddings, segment embeddings and the learned positional encoding, as illustrated in Fig. 2.6.

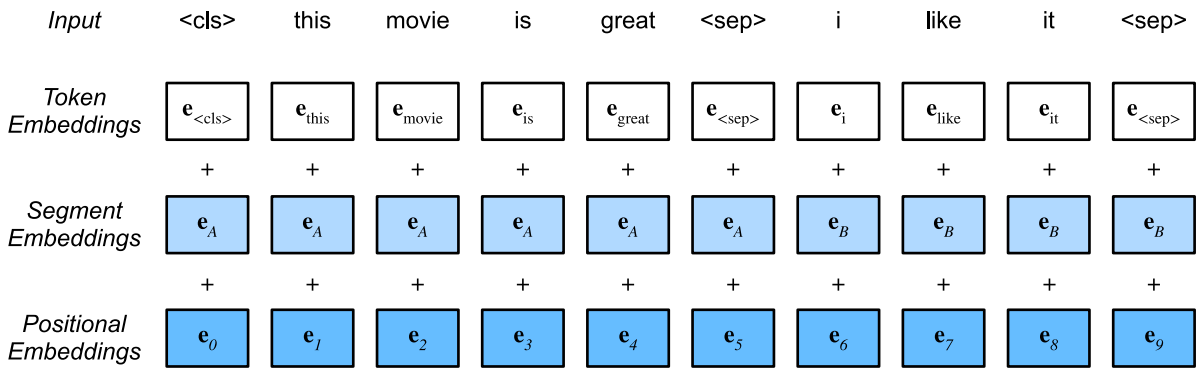

Figure 2.6: The representation of an input sequence for BERT is the sum of the token embeddings, segment embeddings, and positional embeddings. Source: Dive into Deep Learning

2.3.3 *Architecture*

Devlin et al. (2018) use a multi-layer bidirectional Transformer encoder for their network architecture. Furthermore, they present two network sizes, as defined by their number of layers L (N in Vaswani et al. (2017)), hidden size H (d_{model}) and number of self-attention heads A (h).

For both network sizes, a feed-forward size of $4H$ is used, which we will refer to as FF in this work (d_{ff} in Vaswani et al. (2017)). Lastly, in this work we denote the (total) number of parameters of a particular NN as $\#P$. As such, the two network sizes as presented in Devlin et al. (2018) can be represented as follows:

1. $\text{BERT}_{\text{BASE}}$ ($L = 12$, $H = 768$, $FF = 3072$, $A = 12$, $\#P = 110\,\text{M}$)

2. $\text{BERT}_{\text{LARGE}}$ ($L = 24$, $H = 1024$, $FF = 4096$, $A = 16$, $\#P = 340\,\text{M}$)

The former network size is used in the present research as our Transformer-based architecture of choice, as mentioned previously.

2.4 *Knowledge Distillation*

Generally speaking, in the context of NNs, compression is an umbrella term that covers methods that reduce the size-on-disk, methods that reduce the memory footprint, and methods that increase the computational efficiency of an NN (or a combination of these three). Research into such methods has a long history with early, pioneering works dating back to the late 1980s (Janowsky, 1989; Mozer and Smolensky, 1989) and early 1990's (LeCun et al., 1990; Hassibi and Stork, 1993).

Since then, many different techniques have been proposed to make NNs more efficient, which can generally be categorized into three categories: *Pruning*, *Quantization* and *Knowledge Distillation* (KD)[2], the last of which is the focus of this section (and more generally: this thesis).

[2] More recently, a new, fourth category of NN compression techniques has been proposed, called *Theseus Compression* (Xu et al., 2020), named after the famous thought experiment.

KD as introduced by Buciluǎ et al. (2006), and popularized by Hinton et al. (2015), is a NN compression method in which a smaller (untrained) student network is trained to mimic a larger, (pre-)trained teacher network or ensemble of teacher networks by means of *knowledge transfer*. More formally, the student network is trained on some transfer dataset $\mathcal{D}_T$ to minimize a loss function in which the target is the distribution of class probabilities as output by the teacher network ("soft target loss" or "distillation loss"):

$$\mathcal{L}\left(z_s, z_t\right) = KL\left(\sigma\left(z_s\right) || \sigma\left(z_t\right)\right), \tag{2.21}$$

where z_s and z_t are the student and teacher logits (i.e. the raw, unnormalized predictions by the NN), respectively, $\sigma\left(\cdot\right)$ is the softmax function, and $KL(p||q)$ is the *Kullback-Leibler divergence* (KL).

Not only does this soft target loss allow for the data in $\mathcal{D}_T$ to be entirely unlabeled, but it also provides a richer signal during training when compared to only using the ground truth label as your target.

A well-trained teacher network, however, often-times predicts the "true" class (ground truth label) at a (very) high probability, while the other class probabilities are close to 0. This would make the signal no more informative than the ground truth label itself. To address this issue, Hinton et al. (2015) use a *temperature* $T > 1$ in the softmax function (during training) in order to yield a softer probability distribution over the classes:

$$\mathcal{L}_{\text{soft}} = \mathcal{L}\left(z_s, z_t\right) = KL\left(\sigma\left(z_s / T\right) || \sigma\left(z_t / T\right)\right). \tag{2.22}$$

When some or all of the data in $\mathcal{D}_T$ is labeled, Hinton et al. (2015) have found it beneficial, in addition to the soft target loss, to train the student network to also predict the ground truth labels $\boldsymbol{y}$ ("hard target loss" or "student loss"):

$$\mathcal{L}_{\text{hard}} = \mathcal{L}\left(\boldsymbol{z}_s, \boldsymbol{y}\right) = \mathcal{H}\left(\sigma\left(\boldsymbol{z}_s\right), \boldsymbol{y}\right), \tag{2.23}$$

where $\mathcal{H}\left(\sigma\left(\boldsymbol{z}_s\right), \boldsymbol{y}\right)$ is the *Cross Entropy* (CE) between the class probability distribution of the student network and the ground truth labels. Combining the loss functions defined in Eqs. (2.22) and (2.23) yields the following combined loss function:

$$\begin{aligned}\mathcal{L}_{\text{combined}} &= \mathcal{L}\left(\boldsymbol{z}_s, \boldsymbol{z}_t, \boldsymbol{y}\right), \\ &= \alpha \cdot T^2 \cdot KL\left(\sigma\left(\boldsymbol{z}_s / T\right) || \sigma\left(\boldsymbol{z}_t / T\right)\right) \\ &\quad + \beta \cdot \mathcal{H}\left(\sigma\left(\boldsymbol{z}_s\right), \boldsymbol{y}\right),\end{aligned} \tag{2.24}$$

where α and β are coefficients. When using a combination of both soft and hard targets, it is important to multiply the soft target loss by T^2, as the magnitude of its gradient scales as $1/T^2$. Furthermore, in their work, Hinton et al. (2015) found it better to use a simple weighted average of the two objective functions. Practically this means: $\alpha = 0.5$, and $\beta = 1 - \alpha = 0.5$.

2.5 Evaluation

In the context of NLU, certain evaluation methods have significantly risen in popularity in recent years, to the point where one could consider them a "standard". Doing our best not to reinvent the wheel, this work follows the good example set by those before us (see Chapter 3), and uses these "standard" evaluation methods to verify the validity of our approach and compare our approach to others.

To this end, this section first details the *General Language Understanding Evaluation* (GLUE) benchmark in Section 2.5.1, before introducing the *Stanford Question Answering Dataset* (SQuAD) in Section 2.5.2.

2.5.1 GLUE

The GLUE benchmark (Wang et al., 2018) is a diverse collection of 9 NLU tasks (with accompanying datasets) spanning 3 different categories: single-sentence tasks, similarity and paraphrase tasks, and inference tasks.[3] The goal of the benchmark is to assess how well a NN is able to generalize when it comes to NLU (i.e. good performance on the benchmark requires shared knowledge between the different tasks).

[3] The GLUE leaderboard can be found here: https://gluebenchmark.com/leaderboard

A brief description and some relevant statistics of all tasks included in GLUE are presented in Table 2.1, before providing a more detailed description of each task below.

Dataset	|Train|	|Test|	Task	Metric(s)
Single-sentence tasks				
CoLA	3.7 k	1.7 k	acceptability	Matthews correlation coefficient
SST-2	67 k	1.8 k	sentiment	accuracy
Similarity and paraphrase tasks				
MRPC	3.7 k	1.7 k	paraphrase	accuracy/ F_1 score
QQP	364 k	391 k	paraphrase	accuracy/ F_1 score
STS-B	7 k	1.4 k	sentence similarity	Pearson/Spearman correlation coefficients
Inference tasks				
MNLI	393 k	20 k	NLI	matched accuracy / mismatched accuracy
QNLI	105 k	5.4 k	QA / NLI	accuracy
RTE	2.5 k	3 k	NLI	accuracy
WNLI	634	146	coreference / NLI	accuracy

Table 2.1: Descriptions and statistics for all 9 tasks included in the GLUE benchmark. Almost all tasks are binary classification tasks, except for STS-B, which is a regression task, and MNLI, which uses 3 classes. Adapted from Wang et al. (2018).

Single-sentence tasks

CoLA The *Corpus of Linguistic Acceptability* (CoLA) (Warstadt et al., 2019) is a binary classification task, where each sentence is classified as either a grammatical English sentence or not (linguistic acceptability). The sentences were drawn from books and articles on linguistic theory. Performance is evaluated by means of the Matthews correlation coefficient (Matthews, 1975), which gives a score in $[-1, 1]$, where 0 would be akin to random guessing.

SST-2 The *Stanford Sentiment Treebank* (SST) (Socher et al., 2013) is also a binary classification task, where each sentence is classified as either positive or negative (sentiment analysis). The sentences were drawn from movie reviews, with human annotations of their sentiment. Simple classification accuracy is used for evaluation.

Similarity and paraphrase tasks

MRPC The *Microsoft Research Paraphrase Corpus* (MRPC) (Dolan and Brockett, 2005) consists of sentence pairs extracted from online news sources (automatically), with human annotations of whether the sentences are semantically equivalent or not. As the classes are imbalanced (68 % positive), both accuracy and F_1 score are used to measure performance.

QQP *Quora Question Pairs* (QQP) (Iyer et al., 2018) consists of sentence pairs drawn from the popular community QA website Quora, where each sentence pair is classified to be semantically equivalent or not. Similar to MRPC, due to class imbalance (37 % positive), both accuracy and F_1 score are used to measure performance.

STS-B The *Semantic Textual Similarity Benchmark* (STS-B) (Cer et al., 2017) is a regression task, where each sentence pair is classified using scores in $[0, 5]$ to denote their similarity in terms of semantics,

where a score of 0 would mean that the two sentences are completely dissimilar and a score of 5 would mean that the two sentences are completely equivalent. The sentence pairs were drawn from news headlines and other sources, with human annotations for their semantic similarity. Performance is measured by means of Pearson and Spearman correlation coefficients.

Inference tasks

MNLI *Multi-Genre NLI* (MNLI) (Williams et al., 2017) is a textual entailment classification task, where the goal for each sentence pair is to predict whether the first sentence (*premise*) entails the second sentence (*hypothesis*), contradicts it or neither. As such, this is a ternary classification task with *entailment*, *contradiction*, and *neutral* as its labels. The sentence pairs were collected from ten different genres of written and spoken English, such as face-to-face conversations, political speeches, letters, magazine articles, works of fictions, etc. The task actually contains two *sections*: a *matched* section, where the genre of the sentence pairs in the train set match with those in the test set, and a *mismatched* section, where the sentence pairs of the train and test set are of different genres. Performance is measured by accuracy on the matched and mismatched sections (separately).

QNLI *Question-answering NLI* (QNLI) is a recasting of the SQuAD dataset (see Section 2.5.2), where for each question-paragraph pair, question-sentence pairs are formed between the question and each sentence in the paragraph with enough lexical overlap. The task is to classify whether the paragraph sentence contains the answer to the question. Performance is measured using simple classification accuracy.

RTE *Recognizing Textual Entailment* (RTE) is another recasting of existing datasets, where the authors combine the data from RTE1 (Dagan et al., 2005), RTE2 (Haim et al., 2006), RTE3 (Giampiccolo et al., 2007) and RTE5 (Bentivogli et al., 2009) into a single dataset. The dataset consists of sentence pairs with binary textual entailment labels[4]. Classification accuracy is used to evaluate performance.

[4] For RTE5, `UNKNOWN` and `CONTRADICTION` are collapsed to `NOT_ENTAILMENT` for consistency.

WNLI *Winograd NLI* (WNLI) consists of sentence pairs, where the first sentence (*premise*) contains an ambiguous pronoun and the second sentence contains a possible referent (*hypothesis*). The task is to predict whether the first sentence entails the second. The sentence pairs were constructed from the Winograd Schema Challenge dataset (Levesque et al., 2012), which consists of sentences with ambiguous pronouns and a list of possible referents of that pronoun. Performance is measured using classification accuracy.[5]

[5] The authors note that this task has an adversarial development set (examples in the development set sometimes share its hypothesis with an example in the training set), and an imbalanced test set (65 % not entailment), both of which could yield unexpected (negative) results.

2.5.2 *SQuAD*

SQuAD (v1.1) (Rajpurkar et al., 2016) is a sizable reading comprehension dataset with more than 100k crowd-sourced question-passage pairs[6], where the passage contains the answer to the question[7], and the task is to predict the span of text in the passage that contains this answer.[8]

[6] The passages were sourced from articles of English Wikipedia.

[7] Actually, the passage might contain more than one possible answer to the specific question, as we will see in the example provided below.

[8] The SQuAD leaderboard can be found here: https://rajpurkar.github.io/SQuAD-explorer/

Performance is measured using two metrics, which both ignore punctuation marks and articles (e.g. "an", "the", etc.):

Exact Match The *Exact Match* (EM) metric is measured by the number of predictions that match the ground truth answer(s) *exactly*, as a percentage of the total number of predictions.

(Macro-averaged) F_1 score In the context of SQuAD, for a specific question, the F_1 score is computed using the *Bag-of-Tokens* (BoT)[9] representation of both the predicted answer and the ground truth answer. If a question contains multiple ground truth answers, the F_1 score for that question is given by the maximum F_1 score over all the ground truth answers. These F_1 scores for all questions are averaged in order to obtain the macro-average F_1 score.

[9] Analogues to *Bag-of-Words* (BoW), but then for tokens.

An example from the development set of SQuAD is given below, where the different ground truth answers have been color-coded for the convenience of the reader:

Passage:	*Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50.*
Question:	Where did Super Bowl 50 take place?
Ground truth answer(s):	Levi's Stadium Santa Clara, California Levi's Stadium in the San Francisco Bay Area at Santa Clara, California

3 Related Work

Having established a certain understanding of the core concepts that are used throughout this thesis, this chapter presents the reader with an overview of work related to the present research. That is to say: research that has to do with *Knowledge Distillation* (KD) applied to Transformer(-based) architectures.

In and on itself, KD as described by Hinton et al. (2015) provides only a general recipe that leaves much to be specified, such as:

- What (ensemble of) teacher (and student) network architecture(s) do you use?
- What transfer dataset do you use?
- What composition of the combined loss function (see Eq. (2.24)) do you use?

While this provides already plenty degrees of freedom, when it comes to Transformer(-based) architectures specifically, at least one more important variable can be added to the mix:

- At what stage (pre-training, fine-tuning or both) is KD applied?

The last variable to consider is more generally applicable to all scientific research:

- How do you evaluate your approach / method?

As there are (too) many possible categorizations possible along the aforementioned dimensions, this chapter presents the related works chronologically, instead, by date of (pre-)publication. This does mean that, unlike the other chapters in this thesis, the current chapter lacks structure with sections and subsections and soforth. Despite this, it should still be a clear read.

To the best of our knowledge, Tang et al. (2019) were the first to apply (a form of) KD to BERT. Specifically, they applied KD on the end-task level for three different end-tasks (SST-2, MNLI and QQP, a subset of the *General Language Understanding Evaluation* (GLUE) benchmark, see Section 2.5.1). For each end-task, they used $\text{BERT}_{\text{LARGE}}$ fine-tuned on that specific end-task as their teacher network, and a single-layer *Bidirectional LSTM* (BiLSTM) (Hochreiter and Schmidhuber, 1997) as their student network for single-sentence tasks, and its siamese counter part for sentence-pair tasks.

Furthermore, instead of using the *Kullback-Leibler divergence* (KL) of the class probability distributions of the student and teacher net-

works as a soft target loss, Tang et al. (2019) opted to penalize the *Mean Squared Error* (MSE) between the student network's logits against those of the teacher network instead:

$$\mathcal{L}\left(z_s, z_t\right) = \|z_t - z_s\|_2^2 \tag{3.1}$$

Much like the original work of Buciluǎ et al. (2006), Liu et al. (2019a) apply KD to an ensemble of teacher networks in a *Multi-Task Learning* (MTL) setting in order to generalize the knowledge that is transferred to the student network. The authors use the *Multi-Task Deep Neural Network* (MT-DNN) (Liu et al., 2019b) as their architecture of choice for both their teacher networks and student network[1], so there is actually no "compression" here, as the student network is not any smaller or faster than the teacher network.

[1] Actually, MT-DNN is built on top of $\text{BERT}_{\text{LARGE}}$.

First, they trained 6 MT-DNNs with slightly different hyperparameters, before selecting the top 3. These top-3 networks are then fine-tuned on each of four the end-tasks, to form four end-task-specific ensembles, each consisting of 3 MT-DNN teacher networks that are fine-tuned on the specific end-task itself.

Next they use a single MT-DNN as their student network, and apply KD on the end-task level. With this set-up, the logits of the ensemble of N teacher networks are given by:

$$\mathbf{z}_{\text{ensemble}} = \frac{1}{N}\sum_{n=1}^{N} z_n, \tag{3.2}$$

where z_n are the logits of the n^{th} teacher network. For their soft target loss, Liu et al. (2019a) opt for the *Cross Entropy* (CE) (instead of the KL or MSE we've seen previously):

$$\mathcal{L}_{\text{soft}} = \mathcal{L}\left(z_s, z_{\text{ensemble}}\right) = \mathcal{H}\left(\sigma\left(z_s\right), \sigma\left(z_{\text{ensemble}}\right)\right). \tag{3.3}$$

Their overall loss function is a simple average of the hard target loss (as in Eq. (2.23)) and their soft target loss. They evaluate their approach on the GLUE benchmark.

Similarly, Yang et al. (2019b) apply KD to an ensemble of teacher networks in a *Single-Task Learning* (STL) setting instead. The authors use $\text{BERT}_{\text{BASE}}$ as the architecture for their teacher networks, and a smaller version of it that uses only its first three layers as their student network, which they refer to as their "Single Student Model" ($L = 3$, $H = 768$, $FF = 3072$, $A = 12$, $\#P = 45.9\,\text{M}$). Furthermore, they report using N different teachers in the ensemble[2], which are all fine-tuned on the end-task using different hyperparameters.

[2] Nowhere in their work is the value of N specified.

They use cross-entropy as their hard target loss (as in Eq. (2.23)) and a simple average of the MSE of the logits of the student network as compared to those of the teacher (as in Eq. (3.1)):

$$\mathcal{L} = (1-\alpha)\cdot\mathcal{H}\left(\sigma\left(z_s\right), \boldsymbol{y}\right) + \frac{\alpha}{N}\sum_{n=1}^{N}\|z_n - z_s\|_2^2, \tag{3.4}$$

where α is a coefficient[3].

[3] Similarly, nowhere in their work is the value of α specified.

Their end-task of choice is *Question Answering* (QA). For this, the authors use their own experimental dataset, which they call "DeepQA". This dataset consists of 1M samples, where each sample consists of a question, a passage and a binary label indicating whether the question can be answered by the passage.

Recognizing the saliency of the knowledge encoded in intermediate layers of Transformer-based architectures, Sun et al. (2019) introduce *Patient Knowledge Distillation* (PKD), in which a student network *patiently* learns from multiple intermediate layers of the teacher network for incremental knowledge extraction. In addition to the the soft target loss (as in Eq. (2.22)) and hard target loss (as in Eq. (2.23)), they define an additional loss term as the MSE between the (normalized) hidden states of the student and teacher networks:

$$\mathcal{L}_{\text{hidden state}} = \mathcal{L}\left(\boldsymbol{h}_s, \boldsymbol{h}_t\right) = \sum_{l=1}^{L} \left\| \frac{\boldsymbol{h}_s^{(l)}}{\left\|\boldsymbol{h}_s^{(l)}\right\|_2} - \frac{\boldsymbol{h}_t^{(\text{map}(l))}}{\left\|\boldsymbol{h}_t^{(\text{map}(l))}\right\|_2} \right\|_2^2, \tag{3.5}$$

$$\mathcal{L}_{\text{PKD}} = \alpha \cdot \mathcal{L}_{\text{soft}} + (1-\alpha) \cdot \mathcal{L}_{\text{hard}} + \beta \cdot \mathcal{L}_{\text{hidden state}}, \tag{3.6}$$

where $\boldsymbol{h}_s$ and $\boldsymbol{h}_t$ are the student and teacher hidden states, respectively, L denotes the number of layers in the student network, and $\text{map}(\cdot)$ defines a mapping from layers of the student network to layers of the teacher network.

Furthermore, they use $\text{BERT}_{\text{BASE}}$ as their teacher network (which they refer to as $\text{BERT}_1$2), and two smaller versions of it as their student networks: BERT_3 ($L = 3$, $H = 768$, $FF = 3072$, $A = 12$, $\#P = 45.7\,\text{M}$) and BERT_6 ($L = 6$, $H = 768$, $FF = 3072$, $A = 12$, $\#P = 67.0\,\text{M}$), which are initialized from the first 3 (respectively 6) layers of $\text{BERT}_{\text{BASE}}$. KD is applied at the end-task level for a subset of the GLUE benchmark (see Section 2.5.1), namely SST-2, MRPC, QQP, MNLI, QNLI and RTE.

Lastly, the authors experiment with two different mappings (which they call strategies): *Skip*, where the student learns from every k^{th} layer of the teacher:

$$\text{map}_{\text{skip}}(l) = k \cdot l,$$

and *Last*, where the student learns from the last k layers of the teacher:

$$\text{map}_{\text{last}}(l) = l + M - (k+1),$$

where M denotes the number of layers in the teacher network.

Jiao et al. (2019) leverage not only the knowledge from the hidden states and final prediction layer, but also from the embedding layers and attention heads. To this end, they define the following loss

functions:

$$\mathcal{L}_{\text{attn}} = \frac{1}{A}\sum_{a=1}^{A}\left\|\boldsymbol{A}_s^{(a)} - \boldsymbol{A}_t^{(a)}\right\|_2^2, \tag{3.7}$$

$$\mathcal{L}_{\text{hidn}} = \|\boldsymbol{H}_s - \boldsymbol{H}_t\|_2^2, \tag{3.8}$$

$$\mathcal{L}_{\text{embd}} = \|\boldsymbol{E}_s - \boldsymbol{E}_t\|_2^2, \tag{3.9}$$

$$\mathcal{L}_{\text{pred}} = \mathcal{H}\left(\sigma\left(\boldsymbol{z}_s/t\right), \sigma\left(\boldsymbol{z}_s\right)\right), \tag{3.10}$$

where $\boldsymbol{A}_s^{(a)}$ and $\boldsymbol{A}_t^{(a)}$ are the student and teacher attention matrices corresponding to the a^{th} attention head, respectively, $\boldsymbol{H}_s$ and $\boldsymbol{H}_t$ are the student and teacher hidden states, respectively, $\boldsymbol{E}_s$ and $\boldsymbol{E}_t$ are the student and teacher embeddings, respectively, and t is the temperature (Jiao et al. (2019) use $t = 1$).

Using $l = 0$ as the index of the embedding layer and $l = L + 1$ for the index of the prediction layer, the combined loss function is given by:

$$\mathcal{L}_{\text{model}} = \sum_{l=0}^{L+1}\lambda_l \mathcal{L}_{\text{layer}}\left(S_l, T_{\text{map}(l)}\right), \tag{3.11}$$

$$\mathcal{L}_{\text{layer}}\left(S_l, T_{\text{map}(l)}\right) = \begin{cases} \mathcal{L}_{\text{embd}}\left(S_0, T_0\right) & \text{if } l = 0 \\ \mathcal{L}_{\text{hidn}}\left(S_l, T_{\text{map}(l)}\right) + \mathcal{L}_{\text{attn}}\left(S_l, T_{\text{map}(l)}\right) & \text{if } 0 < l \leq L \\ \mathcal{L}_{\text{pred}}\left(S_{L+1}, T_{M+1}\right) & \text{if } l = L+1 \end{cases}, \tag{3.12}$$

where S refers to the student network, T refers to the teacher network, λ_l is a hyperparameter that represents the importance of the l^{th} layer's distillation.

Furthermore, they introduce a two-stage distillation framework:

1. General Distillation: Using a pre-trained teacher network, apply KD during pre-training, such that the student network becomes a compact, general-purpose pre-trained *Language Model* (LM), which can then be fine-tuned on any given end-task.
2. End-task-specific Distillation: Using a teacher network that is fine-tuned on the given end-task, apply KD during fine-tuning, such that the student network becomes a compact, fine-tuned LM that is specialized in the given end-task.

In their work, they use $\text{BERT}_{\text{BASE}}$ as their teacher network and a "tiny" Transformer which they call TinyBert as their student network ($L = 4$, $H = 312$, $FF = 1200$, $A = 12$, $\#P = 14.5\,\text{M}$). During their "General Distillation", they use a "large-scale text corpus"[4] for their transfer dataset. TinyBERT is further fine-tuned and evaluated on the GLUE benchmark (see Section 2.5.1).

[4] Nowhere in their work is the actual composition of this corpus specified

Turc et al. (2019) separate the KD process from either pre-training or fine-tuning. Instead, they first pre-train their student network (a smaller version of BERT) using the *Masked Language Model* (MLM) and *Next Sentence Prediction* (NSP) objectives before performing end-task-specific KD using only a soft target loss (followed by optional

fine-tuning with a hard target loss). They call their method *Pre-trained Distillation* (PD).

Like Devlin et al. (2018), Turc et al. (2019) use a concatenation of the *Toronto BookCorpus* (TBC) and English Wikipedia as their pre-training corpus. After pre-training, they perform KD using unlabeled data ($\mathcal{D}_T$), after which evaluation is performed using labeled data ($\mathcal{D}_L$) for various tasks.

They performed experiments with 24 different student network sizes, ranging from their $\text{Transformer}_{\text{TINY}}$ ($L = 2$, $H = 128$, $FF = 512$, $A = 2$, $\#P = 4.4\,\text{M}$) to their $\text{Transformer}_{\text{BASE}}$ ($L = 12$, $H = 768$, $FF = 3072$, $A = 12$, $\#P = 110.1\,\text{M}$), which has the exact same architecture and size as $\text{BERT}_{\text{BASE}}$. In their experiments, they used both $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{LARGE}}$ as their teacher network. Their approach was evaluated on a subset of the GLUE benchmark, namely: SST-2, MRPC, QQP, MNLI, QNLI and RTE (see Section 2.5.1).

Sanh et al. (2019) opt for a *cosine embedding loss* in order to leverage the knowledge in the intermediate layers, which uses the cosine distance:

$$\mathcal{L}_{\text{cos}} = \mathcal{L}\left(\boldsymbol{h}_s, \boldsymbol{h}_t\right) = 1 - \cos\left(\boldsymbol{h}_s, \boldsymbol{h}_t\right), \tag{3.13}$$

$$\mathcal{L} = \alpha_1 \cdot \mathcal{L}_{\text{soft}} + \alpha_2 \cdot \mathcal{L}_{\text{hard}} + \alpha_3 \cdot \mathcal{L}_{\text{cos}}, \tag{3.14}$$

where $\alpha_1 = \alpha_2 = \alpha_3 = \frac{1}{3}$.

They use $\text{BERT}_{\text{BASE}}$ as their teacher network, and a smaller version of it as their student network, which they call DistilBERT ($L = 6$, $H = 768$, $FF = 3072$, $A = 12$, $\#P = 66.4\,\text{M}$), which is initialized from the 1^{st}, 3^{rd}, 5^{th}, 8^{th}, 10^{th}, and 12^{th} layers of $\text{BERT}_{\text{BASE}}$, respectively[5].

[5] Nowhere in their work do they specify which exact layers from $\text{BERT}_{\text{BASE}}$ they used for the initialization of their DistilBERT. We only learned this after correspondence with the authors.

The authors apply KD during pre-training with the MLM objective, such that the student network becomes a compact, general-purpose pre-trained LM, which can then be fine-tuned on any given end-task.

They evaluate their DistilBERT on both the GLUE benchmark (see Section 2.5.1) and *Stanford Question Answering Dataset* (SQuAD) (see Section 2.5.2) in order to assess the downstream task performance of their general-purpose pre-trained LM.

Similar to Sanh et al. (2019), Sun et al. (2020) apply KD during pre-training, such that the resulting student network is a compact, "task-agnostic", general-purpose LM, which they call MobileBERT. In their set-up, they used a (significantly) modified version of $\text{BERT}_{\text{LARGE}}$ (which they refer to as "Inverse Bottleneck" $\text{BERT}_{\text{LARGE}}$ or IB-$\text{BERT}_{\text{LARGE}}$ for short) as their teacher network and a deep, but thin copy of it as their student network, which they call MobileBERT.

This is different from most approaches discussed previously, where the student network is effectively a more shallow version of the teacher network (i.e. the same hidden size H, feed-forward size FF and number of self-attention heads A, but a smaller number of layers L). Instead, their teacher and student networks boast the same number of layers, but different hidden and feed-forward sizes.

The authors further define a layer-wise knowledge transfer loss $\mathcal{L}_{KT}^{\ell}$ for every ℓ^{th} layer that is very similar to that of Jiao et al. (2019), which is a linear combination of a hidden layer loss $\mathcal{L}_{\text{hidn}}$ (which they call *Feature Map Transfer* (FMT)) and an attention head loss $\mathcal{L}_{\text{attn}}$ (which they call *Attention Transfer* (AT)).

In fact, the hidden layer loss $\mathcal{L}_{\text{hidn}}$ is exactly the same as in Eq. (3.8), whereas for the attention head loss, Sun et al. (2020) use the KL between the attention heads instead of the MSE:

$$\mathcal{L}_{\text{attn}} = \frac{1}{A} \sum_{a=1}^{A} KL\left(\boldsymbol{A}_s^{(a)} || \boldsymbol{A}_t^{(a)}\right), \tag{3.15}$$

where $\boldsymbol{A}_s^{(a)}$ and $\boldsymbol{A}_t^{(a)}$ are the student and teacher attention matrices corresponding to the a^{th} attention head.

Next to this layer-wise knowledge transfer loss, the authors define their *Pre-training Distillation* (PD) loss as a (sort of) linear combination of the MLM loss $\mathcal{L}_{MLM}$, the NSP loss $\mathcal{L}_{NSP}$ and a KD loss $\mathcal{L}_{KD}$:

$$\mathcal{L}_{PD} = \alpha \cdot \mathcal{L}_{MLM} + (1 - \alpha) \cdot \mathcal{L}_{KD} + \mathcal{L}_{NSP}. \tag{3.16}$$

Following the example of Devlin et al. (2018), the authors use a concatenation of the TBC and English Wikipedia as their pre-training corpus. Like Sanh et al. (2019), Sun et al. (2020) evaluate the downstream task performance of their MobileBERT on both the GLUE benchmark (see Section 2.5.1) and SQuAD (see Section 2.5.2).

As a service to the reader, the various papers discussed in this chapter have been summarized in Table 3.1.

Table 3.1: Summary of related works along the questions posed at the start of this chapter. "FT" stands for fine-tuning, whereas "PT" denotes pre-training, and "EnWiki" refers to English Wikipedia.

Paper	**Teacher network**	**Ensemble?**	**Student network(s)**	**Stage**	**Pre-training dataset**	$\mathcal{L}_{\text{soft}}$	$\mathcal{L}_{\text{hard}}$	$\mathcal{L}_{\text{hidden}}$	**Evaluation**
Tang et al. (2019)	$\text{BERT}_{\text{LARGE}}$	×	BiLSTM	FT	×	MSE	CE	×	Subset of GLUE
Liu et al. (2019a)	MT-DNN	✓	MT-DNN	FT	×	CE	CE	×	GLUE
Yang et al. (2019b)	$\text{BERT}_{\text{BASE}}$	✓	BERT_3	FT	×	KL	CE	×	DeepQA
Sun et al. (2019)	$\text{BERT}_{\text{BASE}}$	×	BERT_3 & BERT_6	FT	×	KL	CE	MSE	Subset of GLUE
Jiao et al. (2019)	$\text{BERT}_{\text{BASE}}$	×	TinyBERT	Both	"large-scale text corpus"	CE	×	MSE	GLUE
Turc et al. (2019)	$\text{BERT}_{\text{BASE}}$ & $\text{BERT}_{\text{LARGE}}$	×	Various sizes	FT	TBC + EnWiki	CE	×	×	Subset of GLUE
Sanh et al. (2019)	$\text{BERT}_{\text{BASE}}$	×	DistilBERT	PT	TBC + EnWiki	KL	CE	Cosine distance	GLUE & SQuAD
Sun et al. (2020)	$\text{IB-BERT}_{\text{LARGE}}$	×	MobileBERT	PT	TBC + EnWiki	KL	CE	MSE	GLUE & SQuAD

4 Experimental Design

Before we get to the results of our experiments (our destination, metaphorically speaking), there is one final "stop" along the way: our experimental design, which is precisely the subject of this chapter.

As such, we first introduce the *Neural Network* (NN) architectures for both our student and teacher networks in Section 4.1, highlighting the differences between them in terms of size and inference time. To close out this section, a note on their implementation is given in Section 4.1.1.

Next, a mention of the compute power used throughout this research (both during pre-training and fine-tuning) is given in Section 4.2, followed by description of our pre-training procedure in Section 4.3. This includes a complete overview of all hyperparameters used in Section 4.3.1, and a description of the data used in Section 4.3.2.

Building upon the extensive description of our evaluation methods in Section 2.5, in Section 4.4 we review our process of fine-tuning and evaluating our student network. Like in the previous section, this also includes an overview of all hyperparameters used in Section 4.4.1.

Finally, a brief overview of the baselines used throughout this research is given in Section 4.5. These baselines serve as a means to compare and contrast the results of our evaluation procedure.

4.1 Architecture

Following the examples of Sun et al. (2019); Sanh et al. (2019), we use $\text{BERT}_{\text{BASE}}$ for our teacher network and a shallower version of it, using 6 layers instead of 12, for our student network, which we refer to as $\text{BERT}_{\text{STUDENT}}$ ($L = 6$, $H = 768$, $A = 12$, $\#P = 67.0\,\text{M}$). $\text{BERT}_{\text{STUDENT}}$ is identical to DistilBERT (Sanh et al., 2019) in terms of architecture, and is initialized from the 1^{st}, 3^{rd}, 5^{th}, 8^{th}, 10^{th}, and 12^{th} layers of $\text{BERT}_{\text{BASE}}$, respectively.

A comparison of $\text{BERT}_{\text{BASE}}$ and $\text{BERT}_{\text{STUDENT}}$ in terms of size and inference time is given in Table 4.1.

Network	$\#P$	Size-on-disk	Inference time
$\text{BERT}_{\text{BASE}}$	109.5 M (1.0×)	417 MB (1.0×)	1.47 s (1.0×)
$\text{BERT}_{\text{STUDENT}}$	66.4 M (1.65×)	253 MB (1.65×)	0.80 s (1.83×)

Table 4.1: Comparison of network size (in $\#P$ and size-on-disk) and inference time (in s) between our teacher network, $\text{BERT}_{\text{BASE}}$, and our student network, $\text{BERT}_{\text{STUDENT}}$.

From Table 4.1, we can already see a significant improvement both in terms of size and inference time, though this does not take into account actual performance (in terms of prediction accuracy). We can also note that while $\text{BERT}_{\text{STUDENT}}$ ($L = 6$) uses exactly half the number of layers of $\text{BERT}_{\text{BASE}}$ ($L = 12$), it does not have exactly half the number of parameters. This is primarily due to the embedding layers, which are of the same size for both $\text{BERT}_{\text{STUDENT}}$ and $\text{BERT}_{\text{BASE}}$, as can be reviewed in Table 4.2.

Network	$\#P_{\text{embedding}}$	$\#P_{\text{encoder}}$
$\text{BERT}_{\text{BASE}}$	23.8 M (1.0×)	85.1 M (1.0×)
$\text{BERT}_{\text{STUDENT}}$	23.8 M (1.0×)	42.5 M (2.0×)

Table 4.2: Comparison of network parameters between our teacher network, $\text{BERT}_{\text{BASE}}$, and our student network, $\text{BERT}_{\text{STUDENT}}$.

From Table 4.2, we do see that the number of parameters of the encoder $\#P_{\text{encoder}}$ is linear in the number of layers L.[1]

[1] Note that for $\text{BERT}_{\text{BASE}}$ $\#P_{\text{embedding}}$ and $\#P_{\text{encoder}}$ do not exactly equal $\#P$ (total). This is because of a pooling layer (omitted from Table 4.2), which contains 0.6 M parameters. Following the example of Sanh et al. (2019), this pooling layer is not included in $\text{BERT}_{\text{STUDENT}}$.

4.1.1 *Implementation*

For both our student and teacher networks, as well as our *Knowledge Distillation* (KD) procedure, we use PyTorch (Paszke et al., 2019) as our deep learning framework of choice. More specifically, for the implementation of our student and teacher networks, we use the increasingly popular Transformers library (Wolf et al., 2019) by Hugging Face. For the implementation of our KD procedure, we have developed and released *PyTorch Distillation*, a small extension library for KD in the PyTorch deep learning framework.

4.2 *Compute*

For both pre-training and fine-tuning, we use nodes of the Lisa computing cluster, which contain 2× Intel Xeon Gold 5118 CPUs, 192 GB RAM and 4× 24 GB Nvidia Titan RTX GPUs each, with a theoretical combined peak performance of about 65 TFLOPS for *32-bit Floating Point* (FP32).[2]

[2] For a full description of the Lisa system, please refer to https://userinfo.surfsara.nl/systems/lisa/description.

4.3 *Pre-training*

For our pre-training of $\text{BERT}_{\text{STUDENT}}$, we adapt *Robustly optimized BERT approach* (RoBERTa), the set of best-practices in pre-training BERT(-based) architectures as put forward by Liu et al. (2019c). That is to say, unlike the pre-training of $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018), $\text{BERT}_{\text{STUDENT}}$ is pre-trained without the *Next Sentence Prediction* (NSP)

objective, on large mini-batches, using *dynamic* masking of tokens in the input sequence.

Like Jiao et al. (2019); Sanh et al. (2019); Sun et al. (2020), we apply KD during pre-training. For our (composite) loss function, we use a combination of the *Masked Language Model* (MLM) objective – the *Cross Entropy* (CE) of the token as predicted by the student network and the ground truth token – for the hard target loss (Eq. (2.23)), and the *Kullback-Leibler divergence* (KL) of the token probability distributions of the student and teacher networks for the soft target loss (Eq. (2.22)). Additionally, in order to leverage the knowledge encoded in intermediate layers, the cosine embedding loss between the hidden states of the student and teacher networks is used as well (Eq. (3.13)). This all results in the following composite loss function:

$$\begin{aligned}\mathcal{L} &= \alpha_{\text{soft}} \cdot \mathcal{L}_{\text{soft}} + \alpha_{\text{hard}} \cdot \mathcal{L}_{\text{hard}} + \alpha_{\cos} \cdot \mathcal{L}_{\cos}, \\ &= \alpha_{\text{soft}} \cdot T^2 \cdot KL\left(\sigma\left(\boldsymbol{z}_s / T\right) || \sigma\left(\boldsymbol{z}_t / T\right)\right) \\ &\quad + \alpha_{\text{hard}} \cdot \mathcal{H}\left(\sigma\left(\boldsymbol{z}_s\right), \boldsymbol{y}\right) \\ &\quad + \alpha_{\cos} \cdot \left(1 - \cos\left(\boldsymbol{h}_s, \boldsymbol{h}_t\right)\right).\end{aligned} \tag{4.1}$$

4.3.1 *Hyperparameters*

A complete overview of all hyperparameters used during pre-training is presented in Table 4.3.

Name / Symbol	**Value**
Teacher network	$\text{BERT}_{\text{BASE}}$
Student network	$\text{BERT}_{\text{STUDENT}}$
α_{soft}	$0.\overline{333}$
α_{hard}	$0.\overline{333}$
$\alpha_{\cos}$	$0.\overline{333}$
Optimizer	Adam (Kingma and Ba, 2014)
α	5×10^{-4}
β_1	0.9
β_2	0.98
ϵ	1×10^{-6}
Learning rate policy	1cycle (Smith, 2018)
Min. learning rate	0
Max. learning rate	5×10^{-4}
Warmup proportion	0.05
Min. sequence length	12
Max. sequence length	512
Batch size (per GPU)	5
Nº epochs	3
Nº gradient accumulation steps	40
Max. gradient norm	5.0

Table 4.3: Overview of pre-training hyperparameters.

4.3.2 *Data*

Following the pre-training procedures of both $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018) and DistilBERT (Sanh et al., 2019), we use a concatenation of (a replica of) the *Toronto BookCorpus* (TBC) dataset (Zhu et al., 2015) and

English Wikipedia for our composite pre-training corpus, which totals a little over 3 B words or 18.4 GB of uncompressed text, as can be reviewed in Table 4.4. Both the TBC dataset and English Wikipedia had to be downloaded and pre-processed before being used during pre-training, as described below.

Toronto BookCorpus dataset While used frequently in pre-training *Language Models* (LMs) (Radford et al., 2018; Devlin et al., 2018; Yang et al., 2019a), the TBC dataset (Zhu et al., 2015) is no longer publicly available for download due to copyright issues.[3] Smashwords, the source of books used in the original TBC dataset, still exists, however, and thus allows for the creation of a replica.

[3] In fact, Devlin et al. (2018) admit as much in their GitHub repository, suggesting the use of books from Project Gutenberg instead.

Following the procedure of Van de Graaf (2019b), we have developed and released *Replicate Toronto BookCorpus*, a collection of scripts that aim to replicate the original TBC dataset (Zhu et al., 2015) as faithfully as possible. It does so in 3 steps:

1. Finding those books that meet the criteria (freely available, written in the English language, containing at least 20k words and available to download in plaintext)
2. Downloading these books
3. Pre-processing these books (tokenizing the sentences, concatenating them into a single file and shuffling the lines)

Following these steps, we were able to create a faithful replica of the TBC dataset that is highly comparable, albeit slightly smaller, to the original in terms of size and other important statistics, as presented in Table 4.4.

Dataset	Size	Nº sentences	Nº words	mean w.p.s.	median w.p.s.
TBC (Zhu et al., 2015)	4.7 GB	74.0 M	984.8 M	13.3	11
TBC replica	5.1 GB	65.4 M	897.1 M	13.7	11
English Wikipedia	13.3 GB	109.8 M	2.1 B	19.5	18
TBC replica + English Wikipedia	18.4 GB	175.2 M	3.0 B	17.3	15

Table 4.4: Comparison of datasets in terms of size and other important statistics. "w.p.s." stands for "words per sentence".

English Wikipedia While used at least equally frequently in LM pre-training, English Wikipedia also requires much pre-processing after downloading before being usable as a text corpus. For this, we follow the procedure of Van de Graaf (2019a): After downloading the latest dump of all pages and articles[4], we extract only the text passages (ignoring any lists, tables and headers)[5], before tokenizing the sentences. Finally, all pages and articles are concatenated into a single text-file, before all lines are shuffled.

As can be reviewed from Table 4.4, this produces a text corpus that contains more than 100M sentences and 2B words. A thing of note is that compared to the TBC dataset and its replica, English Wikipedia contains significantly longer sentences on average.

[4] Available at https://dumps.wikimedia.org/enwiki/latest/ for English Wikipedia.

[5] For this, we make use of the WikiExtractor (Attardi, 2015).

Tokenization In order to prevent bottlenecks during pre-training, we tokenize (i.e. create (sub)word embeddings for) our composite pre-training corpus $\mathcal{D}_{\text{Corpus}}$ before the actual pre-training process, instead of during it (on-the-fly). For this, we make use of WordPiece embeddings (Wu et al., 2016) with the exact same vocabulary (i.e. restricted set of tokens) as used by Devlin et al. (2018) (see Section 2.3). For the remainder of this thesis, this pre-tokenized, composite pre-training corpus will be referred to simply as "our corpus" or $\mathcal{D}_{\text{Corpus}}$.

As an example, three sentences are presented below first in their original form (as a sequence of words and punctuation marks), followed by their tokenized form (as a sequence of tokens):

Words: Many religious theologies do not recognise the "law of Christ".
Tokens: `[CLS]` many religious theo ##logies do not recognise the " law of christ " . `[SEP]`

Words: "I'm still concerned about the number of people.
Tokens: `[CLS]` " i ' m still concerned about the number of people . `[SEP]`

Words: No one will find them unless I take them there."
Tokens: `[CLS]` no one will find them unless i take them there . " `[SEP]`

4.4 *Fine-tuning & Evaluation*

$\text{BERT}_{\text{STUDENT}}$ is evaluated by measuring its performance on various down-stream tasks, such as *Natural Language Inference* (NLI) and *Question Answering* (QA). This would both assess the capability of our model to generalize, as well as its aptitude for *Natural Language Understanding* (NLU). To this end, for a given down-stream performance task, $\text{BERT}_{\text{STUDENT}}$ is first fine-tuned (without additional KD) on the training set of that task, before evaluating its performance on the development set of that task.

We fine-tune and evaluate our model on the *General Language Understanding Evaluation* (GLUE) benchmark (see Section 2.5.1), a diverse collection of NLU datasets, and *Stanford Question Answering Dataset* (SQuAD), a sizable QA dataset (see Section 2.5.2).

For each down-stream performance task, we do 5 independent runs, using 5 different random seeds. The reported results are the median of the 5 runs. For GLUE we also report a macro score in addition to the scores for each individual task, which is the simple average of these individual task scores.

4.4.1 *Hyperparameters*

A complete overview of all hyperparameters used during fine-tuning & evaluation is presented in Table 4.5.

Name / Symbol	Value	
	GLUE	**SQuAD**
Network	$\text{BERT}_{\text{STUDENT}}$	$\text{BERT}_{\text{STUDENT}}$
Optimizer	Adam (Kingma and Ba, 2014)	Adam (Kingma and Ba, 2014)
α	2×10^{-5}	3×10^{-5}
β_1	0.9	0.9
β_2	0.98	0.999
ϵ	1×10^{-8}	1×10^{-8}
Learning rate policy	1cycle (Smith, 2018)	1cycle (Smith, 2018)
Min. learning rate	0	0
Max. learning rate	2×10^{-5}	3×10^{-5}
Warmup proportion	0.05	0.05
Max. sequence length	128	384
Max. query length	N.A.	64
Document stride	N.A.	128
Batch size (per GPU)	8	24
Nº epochs	3	3
Nº gradient accumulation steps	1	12
Max. gradient norm	1.0	1.0

Table 4.5: Overview of fine-tuning hyperparameters.

4.5 *Baselines*

In order to verify the significance of our results, we compare the performance of $\text{BERT}_{\text{STUDENT}}$ to two baselines:

$\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018) ($L = 12$, $H = 768$, $A = 12$, $\#P = 110\,\text{M}$)

DistilBERT (Sanh et al., 2019) ($L = 6$, $H = 768$, $A = 12$, $\#P = 67.0\,\text{M}$)

The first baseline, $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018), is our chosen teacher network (i.e. the NN we are trying to compress), so any result in terms of size, speed or performance should be compared to this network. The second baseline, DistilBERT (Sanh et al., 2019), is most similar to our research out of the related works discussed in Chapter 3. In fact, this research directly builds upon theirs. As such, we compare our results to theirs, in order to assess the effects of the changes in KD procedure proposed in this research.

5 Results

Having provided all the necessary concepts and context in the previous chapters, we can now finally get to grips with the heart of this thesis in this chapter: our experiments and their results.

Section 5.1 covers our first experiment, which deals with the size of the transfer dataset $\mathcal{D}_T$ used in the pre-training stage and its effects on the performance of the pre-trained $\text{BERT}_{\text{STUDENT}}$ on various downstream tasks.

The motivation for this experiment is given in Section 5.1.1, followed by the methods in Section 5.1.2. Next, the results of our first experiment are presented in Section 5.1.3, before wrapping up with a discussion in Section 5.1.4.

Our second experiment has to do with the relationship between the composition of the transfer dataset $\mathcal{D}_T$ used in the pre-training stage and the performance of the pre-trained $\text{BERT}_{\text{STUDENT}}$ on various downstream tasks, which is the subject of Section 5.2.

The structure of Section 5.2 is identical to that of our first experiment, starting off with the motivation in Section 5.2.1. Second, the methods are given in Section 5.2.2, followed by the results in Section 5.2.3. Finally, a discussion to our second experiment is given in Section 5.2.4.

5.1 Size of Transfer Dataset $\mathcal{D}_T$

Our first experiment investigates the relationship between the *size* of the transfer dataset $\mathcal{D}_T$ (i.e. the *amount* of data) used for *Knowledge Distillation* (KD) as applied to $\text{BERT}_{\text{BASE}}$, and the performance of the distilled student network on downstream performance tasks (as described in Section 2.5).

5.1.1 Motivation

While proven highly effective at compressing *Neural Networks* (NNs) with minimal loss in performance, when applied to Transformer networks such as $\text{BERT}_{\text{BASE}}$, the KD process itself can be very costly in

terms of the required compute, and thus also financially and environmentally.

For example, Sanh et al. (2019) report pre-training their DistilBERT network for approximately 90 hours on 8× 16 GB Nvidia Tesla V100 GPUs. This would produce about 80 kgs of CO_2 emissions[1] and cost somewhere between $534.2-$1790.2 to train (Lacoste et al., 2019; Strubell et al., 2019). Using nodes of the Lisa computing cluster (as described in Section 4.2), replicating their study would take approximately 156 hours[2] or almost a full week[3].

[1] CO_2 emissions were estimated using the MachineLearning Impact calculator

[2] A single Nvidia Tesla V100 GPU boasts a theoretical performance of about 14 TFLOPS for *32-bit Floating Point* (FP32). As such:

$$90 \cdot \frac{8 \cdot 14.13}{4 \cdot 16.31} \approx 156 \text{ hours.}$$

[3] As the maximum duration for a "job" on the Lisa computing cluster is 120 hours (5 days), it would technically be impossible for us to fully replicate their study, without workarounds like a checkpoint/restart system.

In terms of compute, the "overhead" (i.e. logging, loading data, initializing teacher and student networks, saving results, etc.) that comes with training NNs in general, and Transformer(-based) architectures specifically, is virtually negligible. As such, this high total compute cost described previously can (almost) entirely be attributed to the size of the (transfer) dataset used to train the NN on. In fact, we have found the relationship between the size of the transfer dataset $\mathcal{D}_T$ and the duration of the pre-training stage to be almost linear, as illustrated in Fig. 5.1.

Like Turc et al. (2019); Sun et al. (2020), Sanh et al. (2019) opted for a concatenation of the *Toronto BookCorpus* (TBC) dataset and English Wikipedia for their transfer dataset, which is the exact same data that BERT (Devlin et al., 2018) (their chosen teacher network) was (pre-)trained on.

While in a KD setting it might seem intuitive to pre-train the student network on the same dataset as the teacher network was pre-trained on (after all: the teacher network is supposed to perform well on this dataset), this is not a strict requirement. In fact, to the best of our knowledge, in the context of Transformer(-based) networks like $\text{BERT}_{\text{BASE}}$, the relationship between the size of the transfer dataset $\mathcal{D}_T$ used for KD and the performance of the distilled student network on downstream performance tasks has not been studied yet.

This begs the question: would it be possible to perform KD on a Transformer(-based) NNs like $\text{BERT}_{\text{BASE}}$ using a smaller transfer dataset $\mathcal{D}_T$ and what are the effects on the knowledge transferred from the teacher network to the student network (as measured by proxy in terms of downstream task performance)?

This gap in knowledge is exactly what this experiment aims to address, by distilling the knowledge of our teacher network $\text{BERT}_{\text{BASE}}$ multiple times into our student network $\text{BERT}_{\text{STUDENT}}$, using a transfer dataset $\mathcal{D}_T$ of various sizes (see Section 5.1.2) and reporting the results on downstream performance tasks (see Section 5.1.3).

5.1.2 *Methods*

For the purposes of assessing the relationship between the size of the transfer dataset used for KD and performance of the distilled student network on downstream performance tasks, we take three random

samples (without replacement) from our corpus $\mathcal{D}_{\text{Corpus}}$ (see Section 4.3.2) using three different sample sizes, resulting in three differently sized transfer datasets $\mathcal{D}_T^N$, where N refers to the sample size used to create that transfer dataset. These three differently sized transfer datasets $\mathcal{D}_T^N$ are then the conditions of the current experiment.

The sample sizes used for the random sampling from our corpus are proportional to its size (which consists of 109.3M *sequences* or 4.0B *tokens*) by different orders of magnitude: we take a first random sample that is 10% of its size (10.9M sequences or 402.6M tokens), a second random sample that is 1% of its size (1.1M sequences or 40.3M tokens), and a third and final random sample that is 0.1% of its size (109.4K sequences or 4.0M tokens), as summarized in Table 5.1.

Dataset	Size	Nº sequences	Nº tokens
$\mathcal{D}_{\text{Corpus}}$	20.0 GB	109.3 M	4.0 B
$\mathcal{D}_T^{10\%}$	2.0 GB	10.9 M	402.6 M
$\mathcal{D}_T^{1\%}$	204.8 MB	1.1 M	40.3 M
$\mathcal{D}_T^{0.1\%}$	20.6 MB	109.4 k	4.0 M

Table 5.1: Comparison between our corpus and our transfer datasets (which are differently sized random samples of our corpus).

Following the pre-training procedure set out in Section 4.3, we were able to distill the knowledge of our teacher network $\text{BERT}_{\text{BASE}}$ into our student network $\text{BERT}_{\text{STUDENT}}$ multiple times using the three different transfer datasets. A plot of the duration (in seconds) of the pre-training stage for each of these transfer datasets is given in Fig. 5.1.

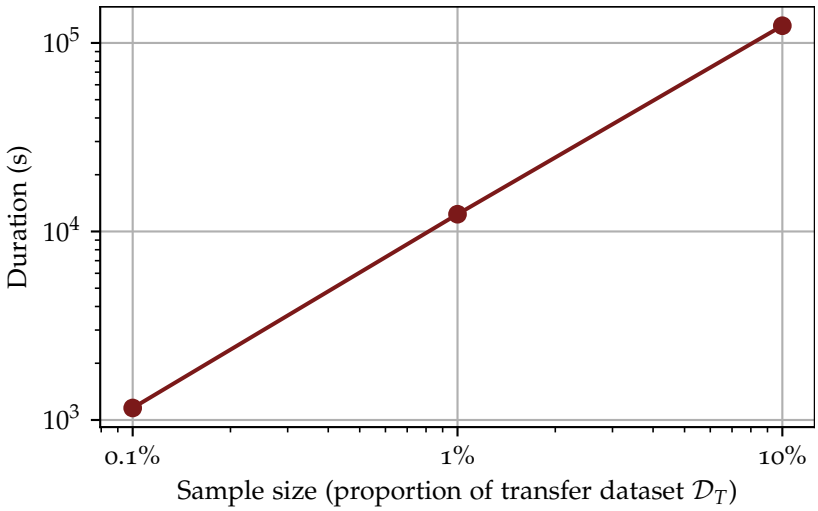

Figure 5.1: A plot of the duration (in seconds) of the pre-training stage for each transfer dataset (referred to by the sample size used to create that transfer dataset).

For all three experimental conditions, $\text{BERT}_{\text{STUDENT}}$ is fine-tuned & evaluated according to the procedure put forward in Section 2.5.

5.1.3 *Results*

GLUE

The performance of our distilled student network $\text{BERT}_{\text{STUDENT}}$ on the *General Language Understanding Evaluation* (GLUE) benchmark for the three experimental conditions are given in Table 5.2. As a reminder to the reader, a description of each GLUE task, along with some relevant statistics are given in Section 2.5.1, specifically Table 2.1.

Network	**Transfer dataset**	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
Ours:											
$\text{BERT}_{\text{STUDENT}}$	$\mathcal{D}_T^{10\%}$	**72.5**	42.6	80.4	76.4	85.8	87.0	56.7	90.3	82.5	50.7
	$\mathcal{D}_T^{1\%}$	**72.4**	37.3	79.6	76.4	86.4	87.2	58.1	89.9	81.7	54.9
	$\mathcal{D}_T^{0.1\%}$	**70.0**	25.2	78.8	75.6	85.0	86.9	59.2	89.1	74.0	56.3
Baselines:											
$\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018)		**74.9**	49.2	80.8	87.4	87.5	86.4	61.7	92.0	83.8	45.1
DistilBERT (Sanh et al., 2019)		**74.3**	43.6	79.0	87.5	85.3	84.9	59.9	90.7	81.2	56.3

Table 5.2: Comparison of performance on the GLUE benchmark using transfer datasets of various sizes. Results of $\text{BERT}_{\text{BASE}}$ and DistilBERT as reported by Sanh et al. (2019).

From Table 5.2 we can infer that $\text{BERT}_{\text{STUDENT}}$ is able to retain 90+% of the performance of its teacher, $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018), even when a transfer dataset is used that is only 0.1 % of the size of the (transfer) dataset as used by Devlin et al. (2018); Sanh et al. (2019). When compared to DistilBERT (Sanh et al., 2019), $\text{BERT}_{\text{STUDENT}}$ achieves 98 % of its performance, despite using a much smaller transfer dataset (as little as 1% proportionally).

Another thing of note is that for some tasks (CoLA and STS-B specifically) the performance of $\text{BERT}_{\text{STUDENT}}$ deteriorates as the amount of data used during pre-training decreases. In fact, when excluding CoLA and STS-B from the computation of the macro score, $\text{BERT}_{\text{STUDENT}}$ is able to retain 96.9+% of the performance of its teacher, $\text{BERT}_{\text{BASE}}$, even for $\mathcal{D}_T^{0.1\%}$.

When less data is used during pre-training, the network has to rely more on the fine-tuning process in order to achieve good downstream task performance. As such, it makes intuitive sense that downstream task performance would decrease as the amount of data used during pre-training decreases, but it does beg the question as to why this is the case only for CoLA and STS-B, and not for the other tasks in the GLUE benchmark.

One possible answer to this question could be that CoLA and STS-B are inherently more difficult than the other tasks in the GLUE benchmark, making them better suited to assess the learned language representations of an NN than the other tasks. Another possible answer could be that there is some sort of error somewhere in the fine-tuning procedure for both CoLA and STS-B that is not present in the fine-tuning procedure for the other tasks, the effects of which become clearer as the network relies more on fine-tuning, rather than pre-

training.

Whatever the reason is that we observe a significant deterioration in downstream task performance only for CoLA and STS-B, it is worth pointing out that while both these tasks contain relatively little training data (3.7k and 7k examples, respectively) as compared to other tasks like QNLI (105k examples), QQP (364k examples) or MNLI (393k examples), this fails to provide a convincing explanation, as other tasks with even less data like WNLI (634 examples), RTE (2.5k examples) and MRPC (3.7k examples) do not experience this drop in performance.

For CoLA, this significant deterioration in task performance could be explained by its inherent class imbalance. Specifically, 70.4 % of the examples in the train split are labeled as linguistically "acceptable" (majority class), whereas only 29.6 % of the examples are labeled as "uncacceptable" (minority class). When $\text{BERT}_{\text{STUDENT}}$ is fine-tuned on the CoLA task without prior pre-training, it simply predicts the majority class 100 % of the time, thus resulting in a score of 0 (akin to random guessing).

For STS-B, the targets are relatively evenly distributed in $[0, 5]$, while the predictions of $\text{BERT}_{\text{STUDENT}}$ are concentrated exclusively in $[0, 0.24]$. As such, it appears that our network suffers from "overfitting" in this case, similar to the CoLA task. A convincing explanation as to why $\text{BERT}_{\text{STUDENT}}$ seems to overfit eludes us yet, however.

Another possible explanation for the decrease in downstream task performance for CoLA and STS-B is that, unlike the other tasks in the GLUE benchmark, the scores for both CoLA and STS-B are determined by means of correlation coefficients (as opposed to classification accuracy). While we cannot convincingly claim this to be the underlying cause, it is worth further investigation, as it could point to an inherent limitation of (smaller) Transformer networks (Vaswani et al., 2017). We leave this for future work.

SQuAD

The performance of our distilled student network $\text{BERT}_{\text{STUDENT}}$ on the *Stanford Question Answering Dataset* (SQuAD) for the three experimental conditions are given are given in Table 5.3.

Network	**Transfer dataset**	**Score**	
		EM	F1
Ours:			
$\text{BERT}_{\text{STUDENT}}$	$\mathcal{D}_T^{10\%}$	56.9	68.5
	$\mathcal{D}_T^{1\%}$	54.7	66.4
	$\mathcal{D}_T^{0.1\%}$	45.0	56.5
Baselines:			
$\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018)		73.9	82.3
DistilBERT (Sanh et al., 2019)		69.6	78.7

Table 5.3: Comparison of performance on SQuAD using transfer datasets of various sizes. Results of $\text{BERT}_{\text{BASE}}$ and DistilBERT as reported by Sanh et al. (2019).

Unlike the (relative) performance of $\text{BERT}_{\text{STUDENT}}$ on GLUE, Table 5.3 shows that on SQuAD there is a more significant loss in performance when less data is used during pre-training.

In the most extreme case, when a transfer dataset is used that is only 0.1 % of the size of the (transfer) dataset as used by Devlin et al. (2018); Sanh et al. (2019), $\text{BERT}_{\text{STUDENT}}$ is able to achieve 68.7 % of the performance of its teacher, $\text{BERT}_{\text{BASE}}$ (Devlin et al., 2018), on average (compared to 80 % for GLUE). For $\mathcal{D}_T^{10\%}$, $\text{BERT}_{\text{STUDENT}}$ retains 83.3 % of the performance of its teacher (compared to 96.8 % for GLUE).

Another interesting observation is that while *Question-answering NLI* (QNLI) is effectively a recasting of SQuAD (see Section 2.5.1, on Page 19), the (relative) performance of $\text{BERT}_{\text{STUDENT}}$ on SQuAD is significantly worse than on QNLI. This would suggest that the recasting of SQuAD not only changed the nature of the task (*Question Answering* (QA) vs. *Natural Language Inference* (NLI)), but also its inherent difficulty.

5.1.4 *Discussion*

A plot of the performance of $\text{BERT}_{\text{STUDENT}}$ on both GLUE and SQuAD combined for the three experimental conditions can be seen in Fig. 5.2.

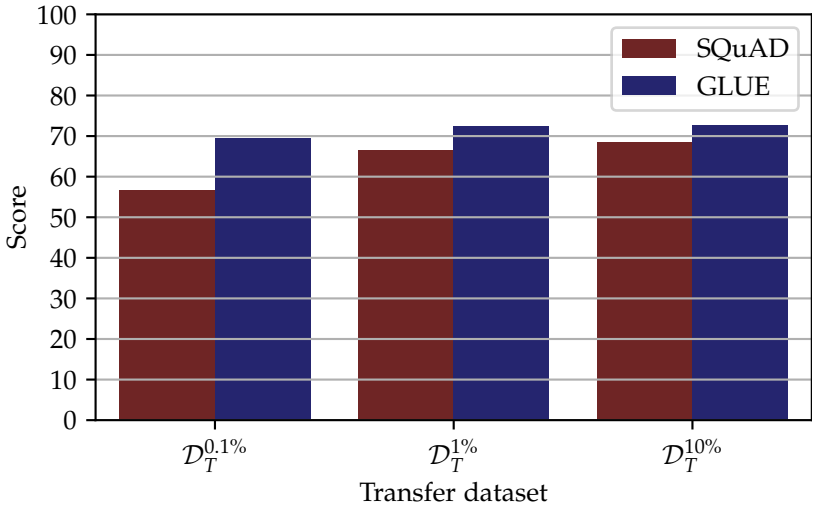

Figure 5.2: Plot of performance of $\text{BERT}_{\text{STUDENT}}$ on GLUE and SQuAD using transfer datasets of various sizes.

From Fig. 5.2, we can see that the performance of $\text{BERT}_{\text{STUDENT}}$ on downstream tasks in general is only marginally worse when pre-trained with a relatively smaller transfer dataset. For certain specific downstream tasks (e.g. MNLI, QNLI, QQP, RTE, SST-2 and WNLI) its performance is even competitive, as can be observed from Tables 5.2 and 5.3. This is both surprising and impressive, and would suggest that significant gains in terms of efficiency come only at a small cost in terms of performance.

Additionally, from Fig. 5.2 we infer that the additional benefit of pre-training with a transfer dataset larger than $\mathcal{D}_T^{0.1\%}$ is marginal for GLUE. Indeed, when $\text{BERT}_{\text{STUDENT}}$ is pre-trained using $\mathcal{D}_T^{10\%}$ as

compared to $\mathcal{D}_T^{0.1\%}$, the increase in performance on GLUE is only 3.6 %. For GLUE, this would suggest that in a KD setting, pre-training with a transfer dataset of "only" $\sim$ 100 k sequences or $\sim$ 4 M tokens is sufficient for good performance.

For SQuAD, the story is pretty similar: we observe diminishing returns in performance when a transfer dataset is used for pre-training that is larger than $\mathcal{D}_T^{1\%}$. From Table 5.3 we can compute that pre-training with a transfer dataset that is 10$\times$ larger than $\mathcal{D}_T^{1\%}$ (i.e. $\mathcal{D}_T^{10\%}$) yields an increase in performance of only 3.1 %. As such, while good performance on SQuAD requires a larger transfer dataset during the pre-training stage as compared to GLUE, there seems to be little added value of using a transfer dataset larger than $\sim$ 1 M sequences or $\sim$ 40 M tokens.

To summarize: we observe diminishing returns in terms of downstream task performance when pre-training with larger transfer datasets. For GLUE, it is sufficient to pre-train with a transfer dataset of "only" $\sim$ 100 k sequences or $\sim$ 4 M tokens for good performance. For SQuAD, there is a benefit to pre-training with a larger transfer dataset, though not larger than $\sim$ 1 M sequences or $\sim$ 40 M tokens. When it comes to the efficiency-performance trade-off, pre-training with huge transfer datasets is clearly sub-optimal.

5.2 *Composition of Transfer Dataset $\mathcal{D}_T$*

From out first experiment (see Section 5.1), we have seen that pre-training with a (much) smaller transfer dataset (i.e. *quantity*) does not interfere much with downstream task performance. In this experiment, we will try to see whether varying the composition of the transfer dataset $\mathcal{D}_T$ (i.e. *quality*) has any (significant) effects on downstream task performance.

5.2.1 *Motivation*

Thus far, we have used only proper *Natural Language* (NL) data for our corpus and derived transfer datasets. "Proper" in this context means that the dataset consists of sentences that are coherent (i.e. have a proper word order), and feature a "natural" composition of words and punctuation marks. Indeed, as described in Section 4.3.2, we use a concatenation of the TBC dataset and English Wikipedia for our corpus, both of which exclusively contain sentences written in English.

The problem with this strategy, however, is that proper NL data is either scarce or requires significant effort in terms of downloading, cleaning and further pre-processing (or both). This is exemplified by our efforts to obtain a corpus that is similar (though not exactly

equal) to the one used by Devlin et al. (2018); Sanh et al. (2019) and others, as described in Section 4.3.2.

When taking languages other than English into consideration, the challenge of creating a corpus of high quantity and quality becomes even more difficult. Taking Afrikaans as an example: the latest dump of Afrikaans Wikipedia (as of May 2021) is an order of magnitude smaller than Dutch Wikipedia and two orders of magnitude smaller than English Wikipedia[4]. Similarly, as of May 2021, Project Gutenberg[5] has only 9 Afrikaans books in its collection, whereas it contains 850 Dutch books and over 50 k books written in English. One final example to further illustrate this point: The latest crawl of Common Crawl[6] (as of May 2021) contains 336.5 k pages in Afrikaans, 58.3 M pages in Dutch, and 1.4 B pages in English.

While using a transfer dataset $\mathcal{D}_T$ consisting of solely proper NL data makes intuitive sense in our context (applying KD to $\text{BERT}_{\text{BASE}}$, which itself was trained solely on NL data), this is not a strict requirement. The primary component of KD is the soft target loss (see Eq. (2.21)), in which the class probability distributions of the student and teacher networks are compared by means of the *Kullback-Leibler divergence* (KL). Theoretically, there is no reason why the outputs of the student and teacher networks cannot be compared for arbitrary input data. As such, it might be possible to use randomized or even generated data for our transfer dataset $\mathcal{D}_T$.

This leads us to pose the following question: would it be possible to perform KD on a Transformer(-based) NNs like $\text{BERT}_{\text{BASE}}$ using a transfer dataset $\mathcal{D}_T$ composed of randomized or even generated data and what are the effects on the knowledge transferred from the teacher network to the student network (as measured by proxy in terms of downstream task performance)?

This experiment attempts to answer precisely this question. To this end, we distill the knowledge of our teacher network $\text{BERT}_{\text{BASE}}$ multiple times into our student network $\text{BERT}_{\text{STUDENT}}$, using a transfer dataset $\mathcal{D}_T$ of different compositions (see Section 5.2.2) and reporting the results on downstream performance tasks (see Section 5.2.3).

[4] As of 05-05-2021, the size of the latest dump of Afrikaans Wikipedia totals 111.28 MB (compressed), whereas the size of the latest dump of Dutch Wikipedia totals 1.55 GB (compressed), and that of English Wikipedia totals 18.53 GB (compressed).

[5] Project Gutenberg is a digital library of over 60000 free e-books, which can be found here: https://www.gutenberg.org/

[6] Common Crawl is an extremely large corpus of crawled web pages, which is available here: https://commoncrawl.org/. Statistics relating to the distribution of languages can be found here: https://commoncrawl.github.io/cc-crawl-statistics/plots/languages

5.2.2 *Methods*

In order to assess the effects of the composition of the transfer dataset used for KD on the performance of the distilled student network on downstream performance tasks, we first create two new transfer datasets of different compositions. These two new transfer datasets are based on $\mathcal{D}_T^{10\%}$ (see Table 5.1), essentially by adding "noise". The newly derived transfer datasets are the conditions of the current experiment.

Randomized data

The first new transfer dataset we create is an randomization of $\mathcal{D}_T^{10\%}$, in which for each sequence of tokens in $\mathcal{D}_T^{10\%}$, the order of the tokens is randomized[7] (i.e. the tokens are shuffled). The pseudo-code for this process is detailed in Algorithm 1.

[7] The positions of the starting classification token `[CLS]` and the ending separation token `[SEP]` remain fixed and are not randomized.

Algorithm 1: Randomize (shuffle) sequences

Input : Sequences array $\boldsymbol{S} = [\, \mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_N \,]$ of size N
Output: Sequences array $\boldsymbol{S}' = [\, \mathbf{s}'_1, \mathbf{s}'_2, \cdots, \mathbf{s}'_N \,]$ of size N

Initialize empty sequences array $\boldsymbol{S}' = [\,\cdot\,]$ of size N;
for $i \leftarrow 1$ **to** N **do**
  Update $\boldsymbol{s} \leftarrow \boldsymbol{S}_i \,(= [\,$ `[CLS]` $t_2\ t_3\ \cdots\ t_{M-2}\ t_{M-1}$ `[SEP]` $\,])$;
  Initialize empty sequence array $\boldsymbol{s}' = [\,\cdot\,]$ of size M;
  Update $\boldsymbol{s}'_1 \leftarrow$ `[CLS]`;
  Update $\boldsymbol{s}'_{2:M-1} \leftarrow shuffle(\boldsymbol{s}_{2:M-1})$;
  Update $\boldsymbol{s}'_M \leftarrow$ `[SEP]`;
  Update $\boldsymbol{S}'_i \leftarrow \boldsymbol{s}'$

Using the example sentences of Section 4.3.2 on Page 32 as an example, their randomized counterparts are presented below:

Original: `[CLS]` many religious theo ##logies do not recognise the " law of christ " . `[SEP]`
Randomized: `[CLS]` of not do theo religious many ##logies . the recognise " christ law " `[SEP]`

Original: `[CLS]` " i ' m still concerned about the number of people . `[SEP]`
Randomized: `[CLS]` m . of i about ' still concerned the " people number `[SEP]`
Original: `[CLS]` no one will find them unless i take them there . " `[SEP]`
Randomized: `[CLS]` " them them will i take find no one . there unless `[SEP]`

As can be reviewed from the examples above, this randomization destroys any information relating to word order[8], but preserves all other properties of $\mathcal{D}_T^{10\%}$ (like sequence length, composition of tokens, etc.). Ultimately, the resulting transfer dataset (labeled as $\mathcal{D}_T^{\text{Rand.}}$) contains sequences that are strictly permutations of the sequences contained in $\mathcal{D}_T^{10\%}$.

[8] Or rather: token order.

Generated data

The second new transfer dataset we create consists of entirely generated data (labeled as $\mathcal{D}_T^{\text{Gen.}}$). For this, we first compute two statistics of $\mathcal{D}_T^{10\%}$, which are then used to generate completely new sequences. This process is further detailed below.

Sequence length The first statistic of $\mathcal{D}_T^{10\%}$ we compute is the frequency at which specific sequence lengths occur, such that for any sequence length l, n_l denotes the number of times this specific sequence length occurs in $\mathcal{D}_T^{10\%}$. These (absolute) sequence length frequencies are then used to compute their relative frequencies, such

that for any sequence length l, its relative frequency f_l is given by:

$$f_l = \frac{n_l}{\sum_l n_l}.$$

A plot of the *cumulative distribution function* (CDF) of these sequence length relative frequencies is presented in Fig. 5.3.

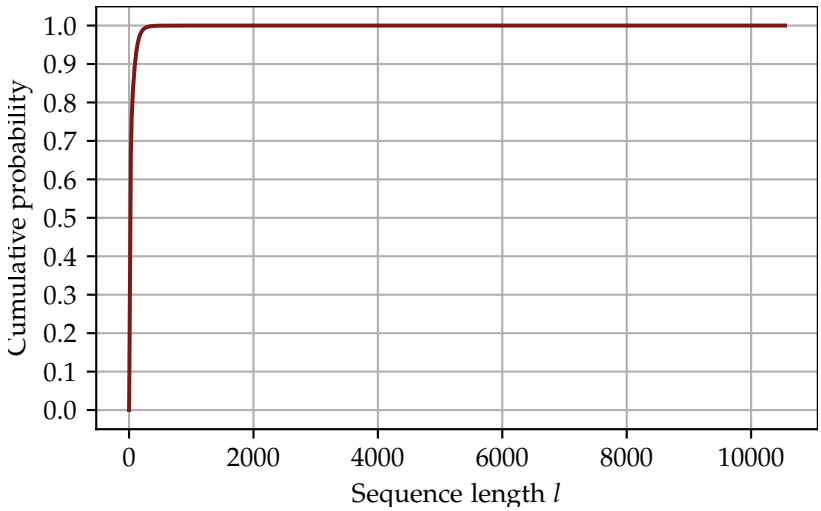

Figure 5.3: A plot of the CDF of the sequence length relative frequencies f_l of $\mathcal{D}_T^{10\%}$. Sequence lengths $l > 400$ are omitted from the plot, as these are all outliers with a relative frequency $f_l \leq 1.115 \times 10^{-5}$.

Token frequencies Similarly, the second statistic of $\mathcal{D}_T^{10\%}$ we compute is the frequency at which specific tokens occur, such that for any token t, n_t denotes the number of times this specific token occurs in $\mathcal{D}_T^{10\%}$. The relative frequency f_t of any token t is given by:

$$f_t = \frac{n_t}{\sum_t n_t}.$$

A plot of the CDF of these token relative frequencies is presented in Fig. 5.4.

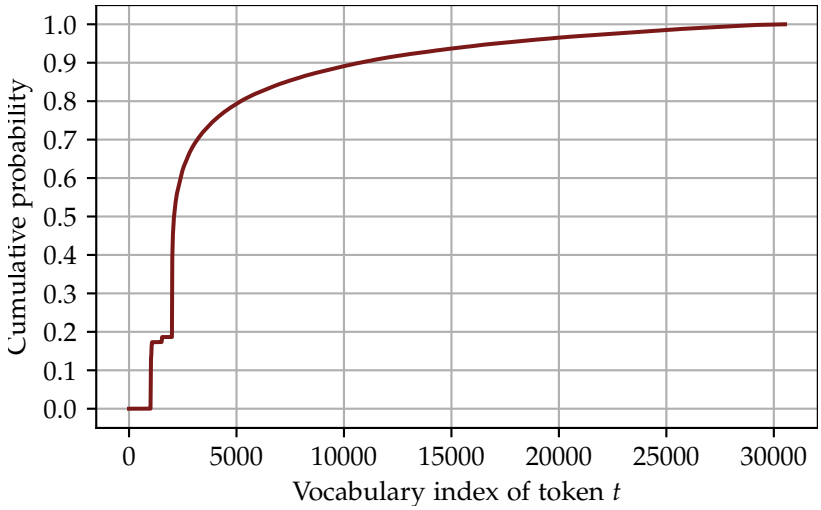

Figure 5.4: A plot of the CDF of the token relative frequencies f_t of $\mathcal{D}_T^{10\%}$.

Combining the sequence length relative frequencies f_l with the token

relative frequencies f_t now allows us to generate new sequences. The pseudo-code for this process is detailed in Algorithm 2.

Algorithm 2: Generate (new) sequences

Input : Number of sequences N to generate
Sequence length relative frequencies f_l
Token relative frequencies f_t
Output: Sequences array $\boldsymbol{S} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \cdots, \boldsymbol{s}_N]$ of size N

Initialize empty sequences array $\boldsymbol{S} = [\cdot]$ of size N;
for $i \leftarrow 1$ **to** N **do**
  Pick a sequence length l with relative frequency f_l;
  Initialize empty sequence array $\boldsymbol{s} = [\cdot]$ of size $l + 2$;
  Update $\boldsymbol{s}_1 \leftarrow$ `[CLS]`;
  for $j \leftarrow 2$ **to** $l + 1$ **do**
    Pick a token t with relative frequency f_t;
    Update $\boldsymbol{s}_j \leftarrow t$;
  Update $\boldsymbol{s}_{l+2} \leftarrow$ `[SEP]`;
  Update $\boldsymbol{S}_i \leftarrow \boldsymbol{s}$;

Some examples of these newly generated sequences are given below:

- `[CLS]` in , the which , . but the gun collection . , dust , `[SEP]`
- `[CLS]`, the , linguistic case a oblique had ram tissue mental which of business , into to for ##rin and `[SEP]`
- `[CLS]` and see or an the ##gr with `[SEP]`

As can be reviewed from the examples above, generating new sequences using the process as detailed in Algorithm 2 destroys any information relating to the proper ordering of tokens. Furthermore, while the tokens are distributed the same (i.e. have the same relative frequencies) in $\mathcal{D}_T^{\text{Gen.}}$ at a macro level, as compared to $\mathcal{D}_T^{10\%}$, when looking at the micro level (i.e. individual sequences), we do observe differences. In other words: the newly generated sequences are not proper NL.

Finally, a comparison of our original transfer dataset $\mathcal{D}_T^{10\%}$, along with the two new transfer datasets $\mathcal{D}_T^{\text{Rand.}}$ and $\mathcal{D}_T^{\text{Gen.}}$, respectively, in terms of size and other relevant statistics is presented in Table 5.4.

Transfer dataset	Size	Nº sequences	Nº tokens
$\mathcal{D}_T^{10\%}$	2.0 GB	10.9 M	402.6 M
$\mathcal{D}_T^{\text{Rand.}}$	2.0 GB	10.9 M	402.6 M
$\mathcal{D}_T^{\text{Gen.}}$	2.0 GB	10.9 M	402.6 M

Table 5.4: Comparison of the transfer datasets used in the current experiment (which are of different compositions).

As can be reviewed from Table 5.4, all three transfer datasets are of the same size. As such, they only differ in their composition (which, again, is our experimental condition).

5.2.3 *Results*

GLUE

The performance of our distilled student network $\text{BERT}_{\text{STUDENT}}$ on the GLUE benchmark for the three transfer datasets are given in Table 5.5.

Transfer dataset	Score	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
$\mathcal{D}_T^{10\%}$	**72.5**	42.6	80.4	76.4	85.8	87.0	56.7	90.3	82.5	50.7
$\mathcal{D}_T^{\text{Rand.}}$	**64.8**	0.0	71.8	76.1	80.9	84.1	54.2	83.8	77.3	54.9
$\mathcal{D}_T^{\text{Gen.}}$	**59.2**	5.2	74.9	76.4	80.9	84.7	56.7	87.6	13.1	53.5

Table 5.5: Comparison of performance on the GLUE benchmark using transfer datasets of various compositions.

Many observations can be made of the results presented in Table 5.5. We will first look exclusively at the macro score, before looking at the scores for the individual tasks contained in the GLUE benchmark.

First, we observe that the performance of $\text{BERT}_{\text{STUDENT}}$ when distilled using either $\mathcal{D}_T^{\text{Rand.}}$ or $\mathcal{D}_T^{\text{Gen.}}$ is worse than when distilled using $\mathcal{D}_T^{10\%}$, though not extremely so. In fact, KD using $\mathcal{D}_T^{\text{Gen.}}$ results in downstream task performance that is only $\sim 20\,\%$ lower (on average) than when using $\mathcal{D}_T^{10\%}$, whereas KD using $\mathcal{D}_T^{\text{Rand.}}$ results in downstream task performance that is only $\sim 10\,\%$ lower (again, on average) than when using $\mathcal{D}_T^{10\%}$. This result is surprising, as one would expect much worse performance on average when pre-training with a less informative[9] transfer dataset.

[9] "Less informative" in this context means that less properties of NL data are preserved, like sequence length, token order, distribution of tokens, etc.

Moreover, we see a 9.4 % higher macro score when using randomized sequences as compared to generated sequences. This result is as expected, as the randomized sequences contained in $\mathcal{D}_T^{\text{Rand.}}$ are more informative than the generated sequences contained in $\mathcal{D}_T^{\text{Gen.}}$.

When looking at the scores for the individual tasks, however, some more nuanced observations and inferences can be made about these results. First, for WNLI we see a 5.5 %+ *increase* in performance when using either $\mathcal{D}_T^{\text{Rand.}}$ or $\mathcal{D}_T^{\text{Gen.}}$ over $\mathcal{D}_T^{10\%}$ during KD. This result is especially surprising, as WNLI is generally considered challenging[10], which gives rise to the expectation that good performance on WNLI requires a NN to be well pre-trained. The challenging nature of WNLI is due to its very small size, imbalanced[11] test set and adversarial development set (see Section 2.5.1 on Page 19).

[10] Devlin et al. (2018) go so far as to call it "problematic".

[11] 65% of the examples in the test set belong the the majority class.

Next, for most tasks we observe a performance for $\mathcal{D}_T^{\text{Rand.}}$ or $\mathcal{D}_T^{\text{Gen.}}$ that is comparable to the performance for $\mathcal{D}_T^{10\%}$, though often times slightly worse. This again is surprising, for the same reasons it was surprising that the macro score was only slightly lower for both $\mathcal{D}_T^{\text{Rand.}}$ and $\mathcal{D}_T^{\text{Gen.}}$, as compared to $\mathcal{D}_T^{10\%}$.

Third and final, we notice significantly worse performance on CoLA and STS-B for both $\mathcal{D}_T^{\text{Rand.}}$ and $\mathcal{D}_T^{\text{Gen.}}$, as compared to $\mathcal{D}_T^{10\%}$, which aligns well with expectations.

In fact, when excluding CoLA and STS-B from the computation of

the macro score, KD using $\mathcal{D}_T^{\text{Rand.}}$ results in downstream task performance that is only $\sim 4.1\,\%$ lower (on average) than when using $\mathcal{D}_T^{10\%}$, whereas KD using $\mathcal{D}_T^{\text{Gen.}}$ results in downstream task performance that is only $\sim 2.4\,\%$ lower (again, on average) than when using $\mathcal{D}_T^{10\%}$. This result is highly similar to our first experiment, where it were also precisely CoLA and STS-B that yielded significantly worse results when $\text{BERT}_{\text{STUDENT}}$ was pre-trained using a smaller transfer dataset. This result underscores the importance of finding a convincing explanation as to why only these tasks are so "difficult" (or analogously: why the other tasks are so "easy").

SQuAD

The performance of our distilled student network $\text{BERT}_{\text{STUDENT}}$ on the SQuAD for the three transfer datasets are given in Table 5.3.

Transfer dataset	**Score**	
	EM	F1
$\mathcal{D}_T^{10\%}$	56.9	68.5
$\mathcal{D}_T^{\text{Rand.}}$	8.7	16.2
$\mathcal{D}_T^{\text{Gen.}}$	27.0	37.3

Table 5.6: Comparison of performance on SQuAD using transfer datasets of various compositions.

Compared to GLUE, Table 5.3 shows a more significant loss in downstream task performance on SQuAD when either $\mathcal{D}_T^{\text{Rand.}}$ or $\mathcal{D}_T^{\text{Gen.}}$ is used during KD, as opposed to when $\mathcal{D}_T^{10\%}$ is used in a KD setting. Specifically, using $\mathcal{D}_T^{\text{Rand.}}$ in a KD setting results in downstream task performance that is $\sim 75\,\%$ lower than when using $\mathcal{D}_T^{10\%}$, whereas KD using $\mathcal{D}_T^{\text{Gen.}}$ results in downstream task performance that is $\sim 45\,\%$ lower than when using $\mathcal{D}_T^{10\%}$. Although this is somewhat similar result as observed for our first experiment, the loss in performance here is significantly worse.

What is also interesting about this result is that the performance on SQuAD when distilling with $\mathcal{D}_T^{\text{Rand.}}$ is significantly worse (56.6 % worse to be precise), than when distilling with $\mathcal{D}_T^{\text{Gen.}}$. Not only is this relationship contrary to the relationship observed for GLUE previously in this experiment, but it is also contrary to our expectations (as mentioned previously).

5.2.4 *Discussion*

A plot of the performance of $\text{BERT}_{\text{STUDENT}}$ on both GLUE and SQuAD combined for the two experimental conditions (three if you include the "original" transfer dataset $\mathcal{D}_T^{10\%}$ as an experimental condition) can be seen in Fig. 5.5.

Fig. 5.5 clearly summarizes our observations and inferences from Tables 5.5 and 5.6: we observe deteriorating performance on GLUE (on average) when $\text{BERT}_{\text{STUDENT}}$ is pre-trained on a transfer dataset that is less informative, though not extremely so. Furthermore, the loss in

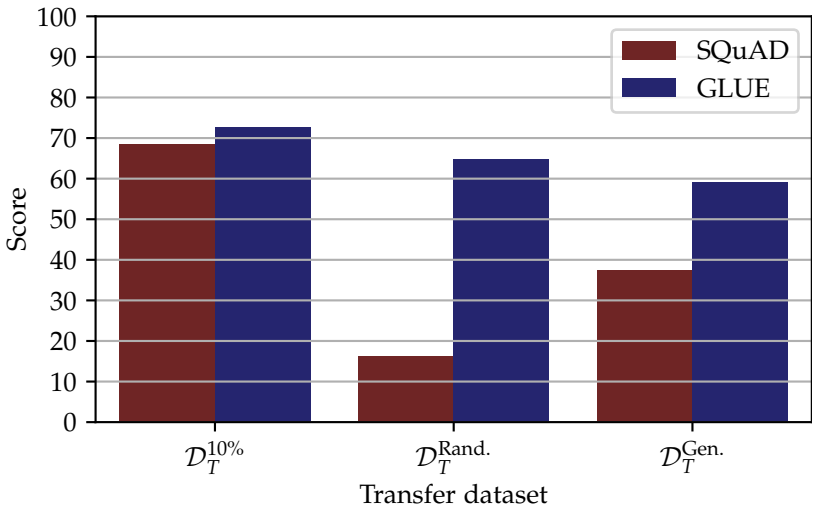

Figure 5.5: Plot of performance of $\text{BERT}_{\text{STUDENT}}$ on GLUE and SQuAD using transfer datasets of various compositions.

performance on SQuAD when using either $\mathcal{D}_T^{\text{Rand.}}$ or $\mathcal{D}_T^{\text{Gen.}}$ (as compared to $\mathcal{D}_T^{10\%}$) in a KD setting is much more significant. This result is also surprising, in that the performance on SQuAD for $\mathcal{D}_T^{\text{Rand.}}$ is significantly worse than for $\mathcal{D}_T^{\text{Gen.}}$. All things considered, however, these results are (much) better than expected, and especially so for GLUE, where for some tasks the downstream performance is even competitive.

One possible explanation for the poor performance observed for CoLA, STS-B and SQuAD (beyond those mentioned for our first experiment), is that our chosen loss function composition (see Eq. (4.1)) is not suited for sequences that disperse with word[12] order. The *Masked Language Model* (MLM) objective (which serves as our hard target loss) specifically does not make sense for unordered sequences. As a reminder to the reader: with the MLM objective, one or more tokens in any given sequence is masked out at random, and the objective entails predicting the original token(s), based only on the context. This objective makes sense and is instructive when your input data consists of ordered sequences, but makes no sense and is uninformative when your sequences are unordered.

[12] or rather: token.

Consider an example of an unordered sequence that contains a single masked token:

Unordered: `[CLS]`. this t a doesn take to `[MASK]` out ' figure it `[SEP]`

Now consider the exact same sequence, but then ordered:

Ordered: `[CLS]` it doesn ' t take a `[MASK]` to figure this out . `[SEP]`

In both cases, the masked word is "genius". Not only is it much more difficult to make a correct prediction for unordered sequences in general, but especially for sequences that are less commonplace (which is the case for most of the sequences contained in our corpus). We leave experimentation with different loss function compositions for future work.

Another limitation of our approach is that it still requires a dataset that consists of proper NL data to derive the new transfer datasets from. This is the case both for randomizing any existing sequences, but also for generating new sequences. While this might not be a problem for high-resource languages (like English and Dutch), it could very well prove problematic for low-resource languages (like Afrikaans).

This limitation also gives rise to additional research questions, such as:

- Would it be possible to "bootstrap" a larger transfer dataset from a smaller one by means of randomizing and/or generating sequences and what would be the effects?
- Would it be possible to create "hybrid" transfer datasets (consisting of both proper NL sequences, randomized and/or generated sequences) and what would be the effects?
- What is the effect of the size of the "original" transfer dataset used for deriving the randomized and/or generated transfer datasets and the quality of these derived transfer datasets (measured by proxy)?

We leave these research questions for future work.

To summarize: while there seems to be little to no benefit to pre-training using a transfer dataset that does not consist of proper NL data in general, there is still much work to be done to draw any definitive conclusions. In this experiment, we tried our best to push the KD process to its proverbial limits by using non-NL data for our transfer dataset. While for CoLA, STS-B and SQuAD this proved to be too much in order to retain good downstream task performance, for other tasks downstream task performance remained competitive.

6 Conclusion

In this thesis, we investigated the relationship between the transfer dataset $\mathcal{D}_T$ used for distilling the knowledge of pre-trained $\text{BERT}_{\text{BASE}}$ into $\text{BERT}_{\text{STUDENT}}$, and the subsequent performance of $\text{BERT}_{\text{STUDENT}}$ on the GLUE benchmark and SQuAD, both in terms of the size of the transfer dataset, as well as its composition. To this end, we performed two primary experiments in hope of answering our initial research questions (see Section 1.2 on Page 4), which are repeated below for the convenience of the reader:

RQ$_1$ What are the effects of the *size* of the transfer dataset $\mathcal{D}_T$ used during pre-training in the Knowledge Distillation process as applied to $\text{BERT}_{\text{BASE}}$, measured in performance on downstream performance task(s)?

RQ$_2$ What are the effects of the *composition* of the transfer dataset $\mathcal{D}_T$ used during pre-training in the Knowledge Distillation as applied to $\text{BERT}_{\text{BASE}}$, measured in performance on downstream performance task(s)?

The results of our two primary experiments can be summarized as follows:

- When it comes to the *size* of the transfer dataset $\mathcal{D}_T$, we observed competitive performance on GLUE and acceptable performance on SQuAD when a transfer dataset is used of "only" $\sim$ 100 k sequences or $\sim$ 4 M tokens. Whereas performance on GLUE does not seem to benefit much from using a larger transfer dataset, performance on SQuAD does.
- With respect to the *composition* of the transfer dataset $\mathcal{D}_T$, we observed diminishing (though still acceptable) performance on GLUE and poor performance on SQuAD when a transfer dataset is used that does not consist of proper NL (i.e. randomized and/or generated) data. More surprisingly, we observe diminishing performance on GLUE when a less "informative" transfer dataset is used for KD, whereas for SQuAD, we observe an increase in performance.

This now allows us to provide answers to our research questions:

- The general effect on downstream task performance of using a smaller transfer dataset $\mathcal{D}_T$ to distill the knowledge of $\text{BERT}_{\text{BASE}}$ into $\text{BERT}_{\text{STUDENT}}$ is a slight decrease in performance[1]. We consider the decrease in performance on GLUE to be marginal, es-

[1] Conversely, the opposite is true for using a larger transfer dataset $\mathcal{D}_T$.

pecially if you also weigh the costs (both financially and environmentally, as well as temporally). Performance on SQuAD benefits significantly more from using a (relatively) larger transfer dataset $\mathcal{D}_T$ during pre-training in the KD process. This answers $\mathbf{RQ}_1$.

- The general effect on downstream task performance of using a transfer dataset $\mathcal{D}_T$ composed of non NL data to distill the knowledge of $\text{BERT}_{\text{BASE}}$ into $\text{BERT}_{\text{STUDENT}}$ is a significant decrease in performance. This is the case for GLUE (though performance is still acceptable), but even more so for SQuAD. This result makes intuitive sense, as you would expect that (pre-)training with less informative data would yield a less informed NN. Nonetheless, this answer is by no means conclusive, as there are still aspects unexplored, which we leave for future work. Moreover, for low-resource languages, randomizing and/or generating sequences might still prove worthwhile. This answers $\mathbf{RQ}_2$.

In the spirit of reproducibility, we publish the source code of this research on GitHub[2]. Likewise, we publish the LaTeX source of this document on GitHub as well[3].

[2] https://github.com/sgraaf/Distilling-BERT

[3] https://github.com/sgraaf/thesis

Finally, we hope this thesis helps advance the research into Knowledge Distillation in general, but especially as applied to Transformer(-based) architectures like $\text{BERT}_{\text{BASE}}$. The promise of this NN compression method is great, as making the latest and greatest Transformer(-based) architectures more efficient (and thereby: more accessible) is imperative.

Bibliography

Attardi, G. (2015). Wikiextractor. https://github.com/attardi/wikiextractor.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bentivogli, L., Clark, P., Dagan, I., and Giampiccolo, D. (2009). The fifth pascal recognizing textual entailment challenge. In *TAC*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Buciluǎ, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541.

Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., and Specia, L. (2017). Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Dagan, I., Glickman, O., and Magnini, B. (2005). The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dolan, W. B. and Brockett, C. (2005). Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*.

Fedus, W., Zoph, B., and Shazeer, N. (2021). Switch transformers:

Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org.

Giampiccolo, D., Magnini, B., Dagan, I., and Dolan, B. (2007). The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics.

van de Graaf, S. (2019a). Pre-processing a wikipedia dump for nlp model training — a write-up. https://towardsdatascience.com/pre-processing-a-wikipedia-dump-for-nlp-model-training-a-write-up-3b9176fdf67. Accessed on 13-12-2019.

van de Graaf, S. (2019b). Replicating the toronto bookcorpus dataset — a write-up. https://towardsdatascience.com/replicating-the-toronto-bookcorpus-dataset-a-write-up-44ea7b87d091. Accessed on 06-12-2019.

Haim, R. B., Dagan, I., Dolan, B., Ferro, L., Giampiccolo, D., Magnini, B., and Szpektor, I. (2006). The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*.

Hassibi, B. and Stork, D. G. (1993). Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

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

Iyer, S., Dandekar, N., and Csernai, K. (2018). Quora question pairs. https://www.quora.com/q/quoradata/First-Quora-Dataset-Release-Question-Pairs/. Accessed on 26-02-2020.

Janowsky, S. A. (1989). Pruning versus clipping in neural networks. *Physical Review A*, 39(12):6600.

Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. (2019). Tinybert: Distilling bert for natural language understanding.

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A., and Kavukcuoglu, K. (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet

classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*.

LeCun, Y., Denker, J. S., and Solla, S. A. (1990). Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605.

Levesque, H., Davis, E., and Morgenstern, L. (2012). The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Liu, X., He, P., Chen, W., and Gao, J. (2019a). Improving multi-task deep neural networks via knowledge distillation for natural language understanding. *arXiv preprint arXiv:1904.09482*.

Liu, X., He, P., Chen, W., and Gao, J. (2019b). Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019c). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451.

Microsoft (2020). Turing-nlg: A 17-billion-parameter language model by microsoft. https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/. Accessed on 09-06-2020.

Mozer, M. C. and Smolensky, P. (1989). Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, pages 107–115.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Ruder, S. (2018). Nlp's imagenet moment has arrived. https://ruder.io/nlp-imagenet/. Accessed on 26-02-2020.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2019). Green AI. *CoRR*, abs/1907.10597.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*.

Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). 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.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Sun, S., Cheng, Y., Gan, Z., and Liu, J. (2019). Patient knowledge distillation for bert model compression. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., and Zhou, D. (2020). Mobilebert: a compact task-agnostic bert for resource-limited devices.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Tang, R., Lu, Y., Liu, L., Mou, L., Vechtomova, O., and Lin, J. (2019). Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*.

Turc, I., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Well-read students learn better: The impact of student initialization on knowledge distillation. *arXiv preprint arXiv:1908.08962*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2018). Glue: A multi-task benchmark and analysis platform for natural language understanding. *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Warstadt, A., Singh, A., and Bowman, S. R. (2019). Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Williams, A., Nangia, N., and Bowman, S. R. (2017). A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. (2019). Huggingface's transformers: State-of-the-art natural language processing.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xu, C., Zhou, W., Ge, T., Wei, F., and Zhou, M. (2020). Bert-of-theseus: Compressing bert by progressive module replacing. *arXiv preprint arXiv:2002.02925*.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019a). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems (NeurIPS)*.

Yang, Z., Shou, L., Gong, M., Lin, W., and Jiang, D. (2019b). Model compression with multi-task knowledge distillation for web-scale question answering system.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and read-

ing books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.