Teacher-student knowledge distillation from BERT

Sam Sučík

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2020

Abstract

Since 2017, natural language processing (NLP) has seen a revolution due to new neural language models – Transformers (Vaswani et al., 2017). Pre-trained on large text corpora and widely applicable even for NLP tasks with little data, Transformer models like BERT (Devlin et al., 2019) became widely used. While powerful, these large models are too computationally expensive and slow for many practical applications. This inspired a lot of recent effort in compressing BERT to make it smaller and faster. One particularly promising approach is knowledge distillation, where the large BERT is used as a "teacher" from which much smaller "student" models "learn".

Today, there is a lot of work on understanding the linguistic skills possessed by BERT, and on compressing the model using knowledge distillation. However, little is known about the learning process itself and about the skills learnt by the student models. I aim to explore both via practical means: By distilling BERT into two architecturally diverse students on diverse NLP tasks, and by subsequently analysing what the students learnt. For analysis, all models are *probed* for different linguistic capabilities (as proposed by Conneau et al. (2018)), and the models' behaviour is inspected in terms of concrete decisions and the confidences with which they are made.

Both students – a down-scaled BERT and a bidirectional LSTM model – are found to learn well, resulting in models up to 14,000x smaller and 1,100x faster than the teacher. However, each NLP task is shown to rely on different linguistic skills and be of different difficulty, thus requiring a different student size and embedding type (word-level embeddings vs sub-word embeddings). On a difficult linguistic acceptability task, both students' learning is hindered by their inability to match the teacher's understanding of semantics. Even where students perform on par with their teacher, they are found to rely on easier cues such as characteristic keywords. Analysing the models' correctness and confidence patterns shows how all models behave similarly on certain tasks and differ on others, with the shallower BiLSTM student better mimicking the teacher's behaviour. Finally, by probing all models, I measure and localise diverse linguistic capabilities. Some possessed language knowledge is found to be merely residual (not necessary), and I demonstrate a novel use of probing for tracing such knowledge back to its origins.

Acknowledgements

I thank Steve Renals of the University of Edinburgh for being incredibly approachable, professional yet friendly. Having had almost 60 weekly meetings with Steve in the past 2 years, I have seen him patiently listening to my updates, providing well-rounded, encouraging feedback and inspiration, helping me to thoroughly enjoy this learning experience.

I am very grateful for being co-supervised by Vladimir Vlasov of Rasa. Throughout the summer internship which inspired this work, as well as later through the academic year, Vova's guidance, endless curiosity and honest opinions helped me to be more ambitious yet realistic, and to critically view the work of others and my own in the particular area.

Many thanks also to Ralph Tang whose work provided a solid starting point for this project, and to Slávka Heželyová who constantly supported me and motivated me to explain my work in non-technical stories and metaphors.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

Natural language processing (NLP) is concerned with using computational techniques to process and analyse human language: for instance, to automatically compute various grammatical properties of a sentence or to analyse its meaning. Since the early 2010s, this area has seen significant improvements due to powerful machine learning methods, especially large artificial neural networks.

In 2017, a new type of neural model was proposed: the Transformer (Vaswani et al., 2017). Since then, numerous Transformer variants were developed (Radford et al., 2018; Devlin et al., 2019; Lan et al., 2019; Liu et al., 2019; Conneau and Lample, 2019) – many of them improving the state-of-the-art results on various NLP tasks[1]. However, these successful models are very large (with hundreds of millions of learnable parameters), which makes them computationally expensive and slow. This limits applications of such models outside of research, in scenarios like real-time sentence processing for human-bot conversations[2].

In an effort to address this downside, a recent stream of research has focused on making Transformers – especially the widely used BERT model (Devlin et al., 2019) – smaller and faster (Michel et al., 2019; Cheong and Daniel, 2019). This includes my own work (Sucik, 2019). Primarily, variations on the *teacher-student knowledge distillation* approach (Bucila et al., 2006) have been used to successfully compress BERT, see Sun et al. (2019b); Mukherjee and Awadallah (2019); Tang et al. (2019b,a); Jiao et al. (2019); Sanh et al. (2019). In knowledge distillation, a large, trained model is used as a *teacher*, and a smaller *student* model learns by observing and trying to mimic the teacher's prediction patterns.

Using knowledge distillation, BERT can be made several times smaller without significant loss of accuracy. While numerous variants of this technique have been successfully developed, there is little understanding of the nature of knowledge distillation: How and what kinds of the large model's knowledge are best learned by the student, and how this

[1] See the leaderboard of the popular GLUE benchmark (Wang et al., 2018) at `gluebenchmark.com/leaderboard`, accessed April 15, 2020.

[2] Take an automated customer support system – a bot. Each customer message gets processed. If the processing model is slow, multiple model instances have to be deployed in order to handle a large number of conversations at once, which in turn requires more resources.

depends on the architecture of the teacher and student models. This gap in understanding is in contrast with the lot of research in understanding the internal properties and linguistic capabilities of BERT (Jawahar et al., 2019; Tenney et al., 2019a; Kovaleva et al., 2019; Lin et al., 2019; Rogers et al., 2020). I argue that it is also important to have a good understanding of knowledge distillation as a tool, and of the smaller and faster models eventually produced by applying this tool to BERT.

1.2 Aims and contributions

In this work, I try to better understand knowledge distillation by exploring its use for knowledge transfer from BERT into architecturally diverse students, on various NLP tasks.

This is further broken down into three aims:

- *Explore the effectiveness of knowledge distillation for very different NLP tasks.* The chosen tasks focus on identifying the sentiment, intent, and linguistic acceptability of single sentences.

  I show that the specific knowledge distillation approach of Tang et al. (2019a) can be used to distil BERT into extremely small students – several thousand times smaller and faster – on two of the NLP tasks. By characterising each task in terms of the linguistic capabilities it requires, I explain the students' inability to match their teacher on the linguistic acceptability task.

- *Explore how distilling knowledge from BERT varies when using different student architectures.* In particular, I use a down-scaled BERT student architecturally similar to the teacher, and a BiLSTM student used previously by Tang et al. (2019b,a), very different from the teacher.

  Both student models are shown to behave similarly. As a novel way of initialising the student models, I use trained sub-word embeddings extracted from the teacher model, and compare them to widely used word embeddings.

- *Explore the linguistic knowledge present in the teacher and how successfully it is learned by the students.* A previously proposed *probing* approach (Conneau et al., 2018) is used for measuring and localising diverse linguistic skills within the models. Secondly, I use a mostly qualitative approach to mine insights from the models' decisions and from the confidence with which the decisions are made.

  I observe that the extent to which the teacher and student models behave similarly depends on the task. Further, for each task, I describe examples which are easy or difficult for the models to classify, and conclude that, in general, the most sophisticated semantic skills are not learnt well by the students. Finally, I show that a model can contain residual language knowledge not needed for the NLP task, and I demonstrate how model probing can help explain the source of such knowledge.

Chapter 2

Background

In this chapter, the Transformer models are introduced and set into the historical context; knowledge distillation is introduced, in particular its recent applications in NLP; and an overview of the most relevant work in model understanding is given.

2.1 NLP before Transformers

By the very nature of natural language, its processing has always meant processing sequences of variable length: be it written phrases or sentences, words (sequences of characters), spoken utterances, sentence pairs, or entire documents. Very often, NLP tasks boil down to making simple decisions about such sequences: classifying sentences based on their intent or language, assigning a score to a document based on its formality, deciding whether two given sentences form a meaningful question-answer pair, or predicting the next word of an unfinished sentence.

As early as 2008, artificial neural networks started playing a key role in NLP: Collobert and Weston (2008)[1] successfully trained a deep neural model to perform a variety of tasks from part-of-speech tagging to semantic role labelling. However, neural machine learning models are typically suited for tasks where the dimensionality of inputs is known and fixed. Thus, it comes as no surprise that NLP research has focused on developing better models that encode variable-length sequences into fixed-length representations. If any sequence (e.g. a sentence) can be embedded as a vector in a fixed-dimensionality space, a simple classification model can be learned on top of these vectors.

One key step in the development of neural sequence encoder models has been the idea of *word embeddings*: rich, dense, fixed-length numerical representations of words. When viewed as a lookup table – one vector per each supported word – such embeddings can be used to "translate" input words into vectors which are then processed further. Mikolov et al. (2013) introduced an efficient and improved way of learning high-quality word embeddings: *word2vec*. The embeddings are learnt as part of the parameters of a larger neural network. The network is forced to learn two tasks: 1) given an incomplete sentence, predicting its next word, and 2) given a word from a sentence, predicting the words

[1] See also Collobert et al. (2011).

preceding the given one in the same sentence[2]. Such training can easily leverage large amounts of unlabelled text data and the embeddings learn to capture various properties from a word's morphology to its semantics. The released word2vec embeddings became very popular due to their easy use and good performance (influential work using word2vec includes Lampele et al. (2016); Kiros et al. (2015); Dos Santos and Gatti (2014); Kusner et al. (2015)).

While word embeddings were a breakthrough, they themselves do not address the issue of encoding a sequence of words into a fixed-size representation. This is where Recurrent neural networks (RNNs) (Rumelhart et al., 1987) come into play. Recurrent models process one word at a time (see Fig. 2.1) while updating an internal ("hidden") fixed-size representation of the text seen so far. Once the entire sequence is processed, the hidden representation (also called "hidden state") can be output and used to make a simple prediction.

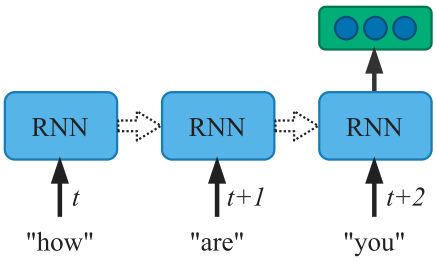


Figure 2.1: A recurrent neural network (RNN) consumes at each timestep one input word. Then, it produces a single vector representation of the inputs.

A common downside of RNNs is that they "forget" over longer sequences. This issue is addressed by introducing learnable *gates*, an idea which soon led to a recurrent model called the Long Short-Term Memory network (LSTM) (Hochreiter and Schmidhuber, 1997). An LSTM unit has a memory cell and learns to selectively add parts of the input into the memory, forget parts of the memory, and output parts of it (see Fig. 2.2). Long after being proposed in 1997, LSTMs gained popularity in NLP – especially in text processing (see e.g. Mikolov et al. (2010) and Graves (2013)).

As various recurrent models started dominating NLP, one particularly influential architecture emerged, addressing tasks such as machine translation, where the output is a new sequence rather than a simple decision. This was the *encoder-decoder* architecture (first described by Hinton and Zemel (1994), later re-introduced in the NLP context by Kalchbrenner and Blunsom (2013) and Sutskever et al. (2014)), see Fig. 2.3. It uses a recurrent encoder to turn an input sentence into a single vector, and a recurrent decoder to generate an output sequence based on the vector.

Bahdanau et al. (2015) improved encoder-decoder models by introducing the concept of *attention*. The attention module helps the decoder produce better output by selectively focusing on the most relevant encoder hidden states at each decoder timestep. This is depicted in Fig. 2.4, showing the decoder just about to output the second word ("estás"). The steps (as numbered in the diagram) are:

[2]These are the so-called Continuous bag-of-words (CBOW) and Skip-gram (SG) tasks, respectively.

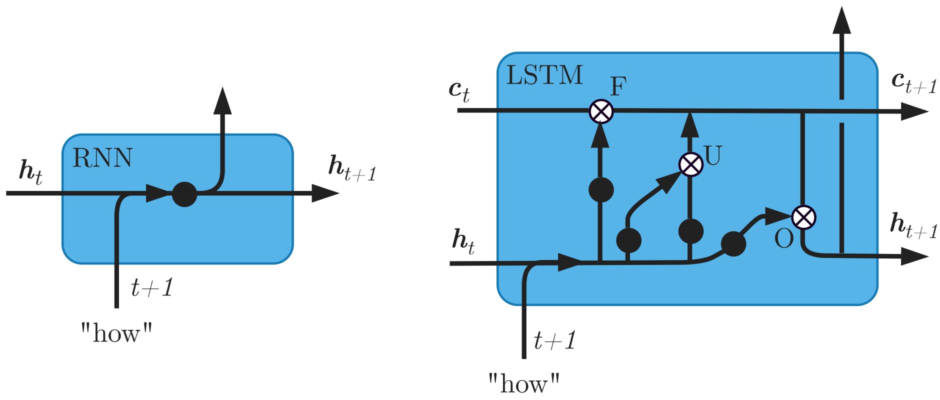


Figure 2.2: Comparing the internals of a vanilla RNN and an LSTM. The latter has three gates (shown as $\otimes$) – the forget gate F, the update gate U, and the output gate O. $\boldsymbol{c}$ is the memory cell, $\boldsymbol{h}$ is the internal (hidden) state which can be used as the output at any timestep. With ● is shown a learnable non-linear transformation.

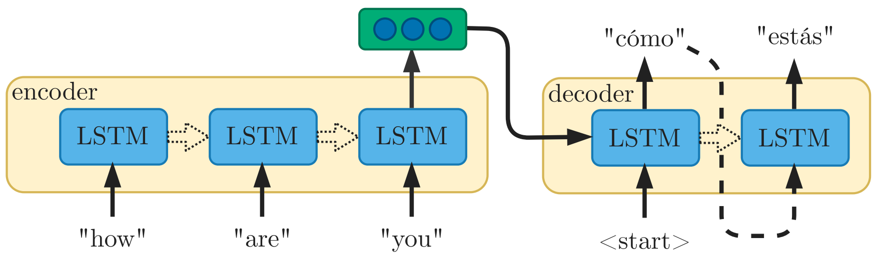


Figure 2.3: An encoder-decoder model for machine translation. Notice how the decoder initially takes as input the special `<start>` token and at later time consumes the previous output word.

1. the decoder's hidden state passed to the attention module,
2. the intermediate hidden states of the encoder also passed to the attention module,
3. the attention module, based on information from the decoder's state, selecting relevant information from the encoder's hidden states and combining it into the attentional *context vector*,
4. the decoder combining the last output word ("cómo") with the context vector and consuming this information to better decide which word to output next.

The attention can be described more formally[3]: First, the decoder state $\boldsymbol{h}_D$ is processed into a *query* $\boldsymbol{q}$ using a learnable weight matrix W_Q:

$$\boldsymbol{q} = \boldsymbol{h}_D W_Q \tag{2.1}$$

and each encoder state $\boldsymbol{h}_E^{(i)}$ (i being the input position or encoder timestep) is used to produce the *key* and *value* vectors, $\boldsymbol{k}^{(i)}$ and $\boldsymbol{v}^{(i)}$:

$$\boldsymbol{k}^{(i)} = \boldsymbol{h}_E^{(i)} W_K, \quad \boldsymbol{v}^{(i)} = \boldsymbol{h}_E^{(i)} W_V \ . \tag{2.2}$$

[3] My description does not exactly follow the original works of Bahdanau et al. (2015) and Luong et al. (2015). Instead, I introduce concepts that will be useful in later sections of this work.

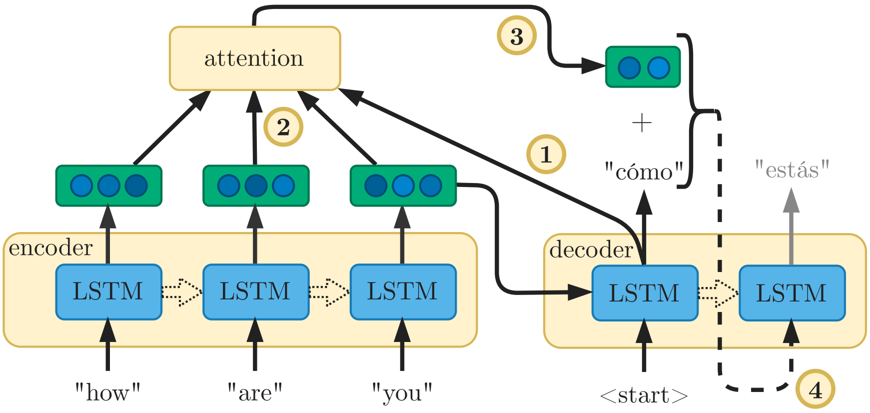


Figure 2.4: An encoder-decoder model for machine translation with added attention mechanism.

Then, the selective focus of the attention is computed as an *attention weight* $w^{(i)}$ for each input position i, by combining the query with the i-th key:

$$w^{(i)} = \boldsymbol{q}^\top \boldsymbol{k}^{(i)} \ . \tag{2.3}$$

The weights are normalised using softmax and used to create the context vector $\boldsymbol{c}$ as a weighted average of the values:

$$\boldsymbol{c} = \sum_i a^{(i)} \boldsymbol{v}^{(i)} \quad \text{where} \quad a^{(i)} = \text{softmax}(w^{(i)}) = \frac{\exp\left(w^{(i)}\right)}{\sum_j \exp\left(w^{(j)}\right)} \ . \tag{2.4}$$

Note that W_Q, W_K, W_V are matrices of learnable parameters, optimised in training the model. This way, the attention's "informed selectivity" improves over time.

For about 4 years, recurrent models with attention were the state of the art in many NLP tasks. However, as we will see, the potential of attention reached far beyond recurrent models.

2.2 Transformer-based NLP

2.2.1 Transformers

We saw how the attention mechanism can selectively focus on parts of a sequence to extract relevant information from it. This raises the question of whether processing the inputs in a sequential fashion with the recurrent encoder is still needed. In particular, RNN models are slow as a result of this sequentiality, and are hard to parallelise. In their influential work, Vaswani et al. (2017) proposed an encoder-decoder model based solely on attention and fully parallelised: the *Transformer*. The core element of the model is the *self-attention* mechanism, used to process all input words in parallel.

In particular, a Transformer model typically has multiple self-attention layers, each layer processing separate representations of all input words. Continuing with the three-word input example from Fig. 2.4, a high-level diagram of the workings of a self-attention layer is shown in Fig. 2.5. Importantly, the input word representations evolve from lower to higher layers such that they consider not just the one input word, but also all other words – the representation becomes *contextual* (also referred to as a *contextual embedding* of the word within the input sentence).

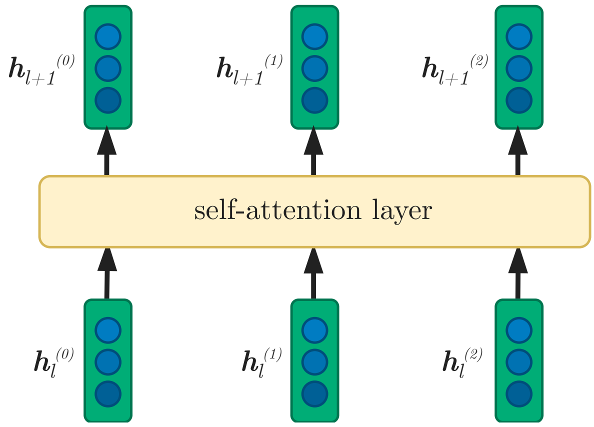


Figure 2.5: A high-level diagram of the application of self-attention in Transformer models. Three hidden states are shown for consistency with the length of the input shown in Fig. 2.4; in general, the input length can vary.

As for the internals of self-attention, the basic principle is very similar to standard attention. Self-attention too is used to focus on and gather relevant information from a sequence of elements, given a query. However, to produce a richer contextual embedding $\boldsymbol{h}_{l+1}^{(i)}$ of the i-th input word in layer $l+1$, self-attention uses the incoming representation $\boldsymbol{h}_l^{(i)}$ for the query, and considers focusing on all representations in layer l, including $\boldsymbol{h}_l^{(i)}$ itself. Fig. 2.6 shows this in detail for input position $i = 1$. Query $\boldsymbol{q}^{(1)}$ is produced and matched with every key in layer l (i.e. $\boldsymbol{k}^{(0)}, \ldots, \boldsymbol{k}^{(2)}$) to produce the attention weights. These weights quantify how relevant each representation $\boldsymbol{h}_l^{(i)}$ is with respect to position $i = 1$. Then, the new contextual embedding $\boldsymbol{h}_{l+1}^{(i)}$ is constructed as a weighted sum of the values $\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(2)}$ (same as constructing the context vector in standard attention).

Notice that, even though each contextual embedding considers all input positions, the next-layer contextual embeddings $\boldsymbol{h}_{l+1}^{(0)}, \ldots, \boldsymbol{h}_{l+1}^{(2)}$ can be computed all at the same time, in parallel: First, the keys, queries and values for all input positions are computed; then, the attention weights with respect to each position are produced; finally, all the new representations are produced. It is this parallelism that allows Transformer models to run faster. As a result, they can be much bigger (and hence create richer input representations) than recurrent models while taking the same time to train.

Due to their parallel nature, self-attentional layers have no notion of an element's position within the input sequence. This means no sensitivity to word order. (Recurrent models sense this order quite naturally because they process input text word by word.) To alleviate this downside of self-attention, Transformers use *positional embeddings*. These are artificially created numerical vectors added to each input word, different across input positions, thus enabling the model's layers to learn to be position- and order-sensitive.

As an additional improvement of the self-attentional mechanism, Vaswani et al. intro-

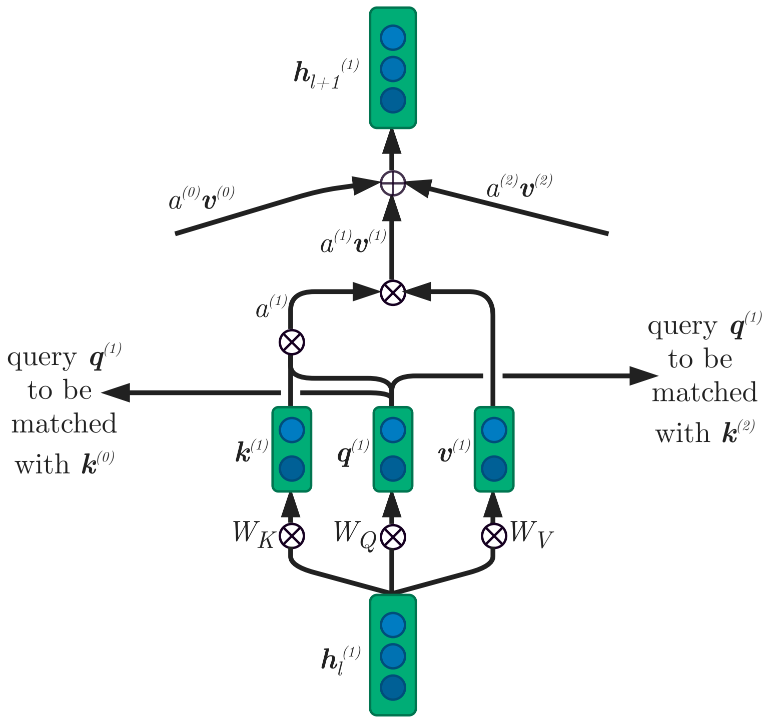


Figure 2.6: The internals of self-attention, illustrated on creating the next-layer hidden representation of the input position $i = 1$, given all representations in the current layer (previous, current, and following). Note that $\otimes$ stands for multiplication (where the multiplication involves a learnable matrix like W_K, this is written next to the $\otimes$), and $\oplus$ denotes summation.

duce the concept of multiple self-attention heads. This is very similar to having multiple instances of the self-attention module in Fig. 2.6, each instance being one *head* and computing its own queries, keys and values. The motivation behind multiple self-attention heads is to enable each head a to learn different "focusing skills" by learning its own $W_{Q,a}$, $W_{K,a}$, $W_{V,a}$. Each head produces its own output[4]:

$$\boldsymbol{O}_{att,a} = \text{softmax}(\frac{\boldsymbol{qk}}{\sqrt{d_k}})\boldsymbol{v} = \text{softmax}(\frac{(\boldsymbol{h}_l^\top W_{Q,a})(\boldsymbol{h}_l^\top W_{K,a})}{\sqrt{d_k}})(\boldsymbol{h}_l^\top W_{V,a}) \tag{2.5}$$

which matches Fig. 2.6 (but notice the detail of the additional scaling by $\frac{1}{\sqrt{d_k}}$, introduced by Vaswani et al., where d_k is the dimensionality of the key). The outputs of the A individual attentional heads are then concatenated and dimensionality-reduced with a trainable linear transformation W_{AO}, to produce the final output, which replaces $\boldsymbol{h}_{l+1}$ in Fig. 2.6:

$$\boldsymbol{O}_{att} = [\boldsymbol{O}_{att,1},\ \ldots,\ \boldsymbol{O}_{att,A}]W_{AO}\ . \tag{2.6}$$

Besides the self-attention-based architecture, there is one more important property that makes today's Transformer models perform so well on a wide variety of NLP tasks: the way these models are trained. First used for Transformers by Radford et al. (2018)[5], the general procedure is:

[4]Here, $\boldsymbol{q}$, $\boldsymbol{k}$, $\boldsymbol{v}$, $\boldsymbol{h}$ are column vectors.

[5]The idea was previously used with recurrent models by Dai and Le (2015).

1. *Unsupervised pre-training*: The model is trained on one or more tasks, typically language modelling, using huge training corpora. For example, Radford et al. pre-train their model to do next word prediction (the standard language modelling task) on a huge corpus of over 7,000 books.

2. *Supervised fine-tuning*: The pre-trained model is trained on a concrete dataset to perform a desired downstream task, such as predicting the sentiment of a sentence, translating between languages, etc.

This two-step procedure is conceptually similar to using pre-trained word embeddings. In both cases, the aim is to learn general language knowledge and then use this as a starting point for focusing on a particular task. However, in this newer case, the word representations learned in pre-training are better tailored to the specific architecture, and they are inherently contextual – compared to pre-trained word embeddings like word2vec which are typically context-insensitive.

Importantly, pre-trained knowledge makes models more suitable for downstream tasks with limited amounts of labelled data. The model no longer needs to acquire all the desired knowledge just from the small dataset; it contains pre-trained high-quality general language knowledge which can be reused in various downstream tasks. This means that large, powerful Transformer models become more accessible: They are successfully applicable to a wider array of smaller tasks than large models that have to be trained from scratch.

2.2.2 BERT

Perhaps the most popular Transformer model today is BERT (Bidirectional Encoder Representations from Transformers), proposed by Devlin et al. (2019). Architecturally, it is a sequence encoder, hence suited for sequence classification tasks. While being heavily based on the original Transformer (Vaswani et al., 2017), BERT also utilises a number of further ideas:

1. The model learns bidirectional representations: It can be trained on language modelling that is not next-word prediction (prediction given left context), but word prediction given both the left and the right context.

2. It uses two very different pre-training classification tasks:

   (a) The *masked language modelling* (MLM) task encourages BERT to learn good contextual word embeddings. The task itself is to correctly predict the token at a given position in a sentence, given that the model can see the entire sentence with the target token(s) masked out[6], replaced with a different token, or left unchanged.

   (b) The *next-sentence prediction* (NSP) task encourages BERT to learn good sentence-level representations. Given two sentences, the task is to predict whether they formed a consecutive sentence pair in the text they came from, or not.

[6] I.e. replaced with the special `[MASK]` token.

The pre-training was carried out on text from books and from the English Wikipedia, totalling to 3,400 million words (for details see Devlin et al. (2019)). The MLM and NSP tasks were both used throughout the pre-training, forcing the model to learn both at the same time.

3. The inputs are processed not word by word, but are broken down using a fixed vocabulary of sub-word units called *wordpieces* (conceptually introduced by Sennrich et al. (2016), this particular variant created by Wu et al. (2016)). This way, BERT can better deal with rare words – by assembling them from pieces[7]. The tokeniser module of BERT uses the wordpiece vocabulary of Wu et al. to tokenise (segment) the input text before it is further processed. Fig. 2.7 shows an example; notice how my surname ("Sucik") gets split into three wordpieces whereas the other, much more common words are found in the wordpiece vocabulary.

4. To enable the different pre-training tasks as well as two-sentence inputs, BERT uses a special input sequence format, illustrated in Fig. 2.7. Given the two input sentences S_A, S_B, they are concatenated and separated by the special `[SEP]` token. The overall sequence is prepended with the `[CLS]` (classification) token. To explicitly capture that certain tokens belong to S_A and others to S_B, simple *token type embeddings* (which only take on two different values) are added to the token embedding at each position. Then, for tasks like NSP, only the output representation of the `[CLS]` token (i.e. $\boldsymbol{o_0}$) is used, whereas for token-level tasks like MLM the output vector from the desired position is used (in Fig. 2.7, the MLM task would use $\boldsymbol{o_3}$ to predict the correct token at this position).

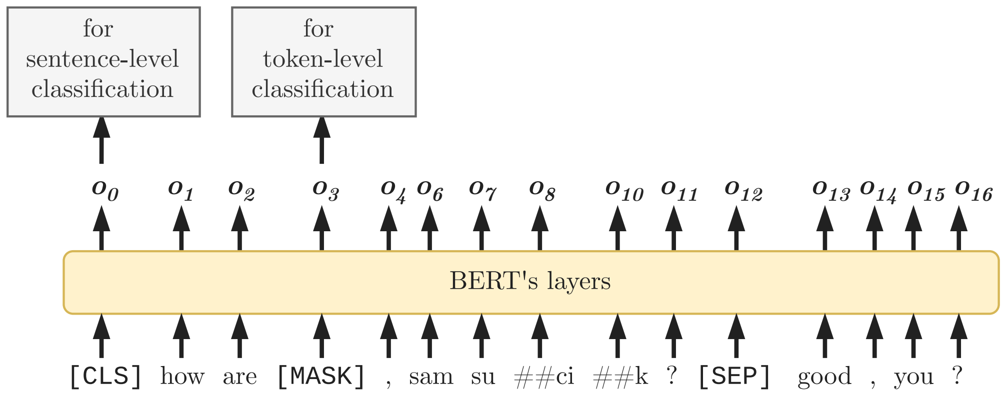


Figure 2.7: BERT's handling of input for sentence-level and token-level tasks. The input sentences ($S_A = \textit{How are you, Sam Sucik?}$ and $S_B = \textit{Good, you?}$) are shown as split by BERT's tokeniser, with the first instance of "you" masked out for MLM.

The overall architecture of BERT is shown in Fig. 2.8. The tokeniser also adds the special tokens like `[CLS]` and `[SEP]` to the input, while the trainable token embedding layer also adds the positional embedding and the token type embedding to the wordpiece embedding of each individual token. The pooler takes the appropriate model output (for sequence level classification the first output $\boldsymbol{o_0}$ as discussed above) and applies a fully-connected layer with the tanh activation function. The external classifier is often another fully-

[7] In word-level models, words that are not found in the model's vocabulary are replaced with a special `UNKNOWN` token, which means disregarding any information carried by the words.

connected layer with the tanh activation, producing the logits[8]. These get normalised using softmax to produce a probability distribution over all classes. The most probable class gets output as the model's prediction.

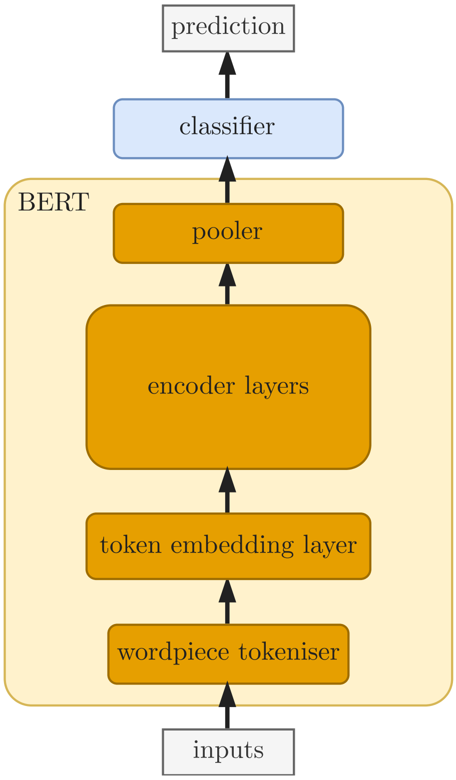


Figure 2.8: High-level overview of the modules that make up the architecture of BERT as used for sequence-level classification.

To complete the picture of BERT, Fig. 2.9 shows the internals of an encoder layer. Besides the multi-headed self-attention submodule, it also contains the fully-connected submodule. This uses a very wide intermediate fully-connected transformation with parameters W_I, inflating the representations up to the dimensionality d_I, and the layer output fully-connected transformation with parameters W_O, which reduces the dimensionality. Each submodule is also by-passed by a residual connection (shown with dashed lines). The residual information is summed with the submodule's output, and layer normalisation is applied to the sum. Note that this structure is not new in BERT; it was used already by the original Transformer of Vaswani et al. (2017). Conveniently, Transformers are designed such that all of the intermediate representations (especially the encoder inputs and outputs, and the self-attention layer inputs and outputs) have the same dimensionality d_h – this makes any residual by-passing and summing easy.

When training BERT, artificial, intentional corruption of internal representations is done using dropout, which acts as a regulariser, making the training more robust. In particular, dropout is applied to the outputs of the embedding layer, to the computed attention weights, just before residual summation both to the self-attention layer output and to the fully connected layer output (see Fig. 2.9 for the summation points), and to the output

[8]For a classifier, the logits are the (unnormalised) predicted class probabilities.

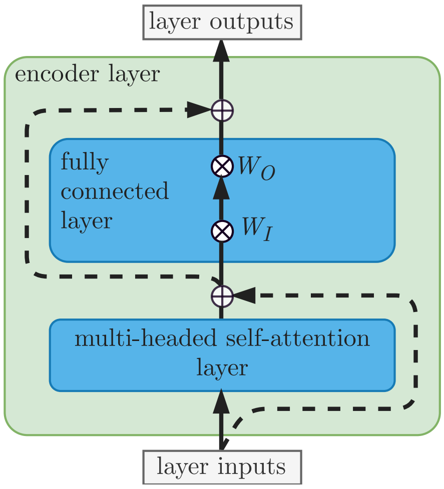


Figure 2.9: The modules making up one encoder layer in BERT; residual connections highlighted by using dashed lines. $\otimes$ marks learnable neural layers, $\oplus$ marks summation (in this case used to combine residual information with layer outputs).

of the pooler module (before applying the external classifier, see Fig. 2.8). The typical dropout rate used is 0.1.

For updating the learnable parameters during training, BERT uses the popular Adam learning algorithm (Kingma and Ba, 2015), which combines two main ideas:

1. *Adaptive learning rates*, meaning that each learnable model parameter can have its own "pace of learning". In Adam, this individual pace is based on the observed recent gradients of the overall model error with respect to the single parameter.

2. *Momentum*, a mechanism used to deal with complex, stochastic error surfaces, by preferring only that direction in the parameter space, which leads to stable improvements (and dispreferring directions which only result in short-term, stochastic improvements). Two decay rates β_1 and β_2 realise the momentum – they control how quickly and noisily or slowly and smoothly the adaptation of the learning rate happens. In practice, the high values $\beta_1 = 0.9$ and $\beta_2 = 0.999$ are often used (as recommended by Kingma and Ba), meaning relatively slow and smooth adaptation.

Originally, pre-trained BERT was released in two sizes: $\text{BERT}_{\text{Base}}$ with 110 million parameters, 12 encoder layers and 12-head self-attention, and $\text{BERT}_{\text{Large}}$ with 340 million parameters, 24 encoder layers and 16-head self-attention. The models quickly became popular, successfully applied to various tasks from document classification (Adhikari et al., 2019) to video captioning (Sun et al., 2019a). Further pre-trained versions were released too, covering, for example, the specific domain of biomedical text (Lee et al., 2019) or multilingual text (Pires et al., 2019).

2.2.3 Newer and larger Transformer models

Following the success of the early Transformers and BERT (Vaswani et al., 2017; Radford et al., 2018; Devlin et al., 2019), many further model variants started emerging, including:

- The OpenAI team releasing GPT-2 (Radford et al., 2019), a larger and improved version of their original, simple Transformer model GPT (Radford et al., 2018).
- Conneau and Lample (2019) introducing XLM, which uses cross-lingual pre-training and is thus better suited for downstream tasks in different languages.
- Transformer-XL (Dai et al., 2019), which features an improved self-attention that can handle very long contexts (across multiple sentences/documents).

All these open-sourced, powerful pre-trained models were a significant step towards more accessible high-quality NLP (in the context of downstream tasks with limited data). However, the model size – often in 100s of million trainable parameters – meant these models could not be applied easily in practice (outside of research): They were memory-hungry and slow.

Naturally, this inspired another stream of research: Compressing large, well-performing Transformer models (very often BERT) to make them faster and resource-efficient. I turn my focus to one compression method that worked particularly well so far: the teacher-student knowledge distillation.

2.3 Teacher-student knowledge distillation

2.3.1 A brief introduction to knowledge distillation

Knowledge distillation was introduced by Bucila et al. (2006) as a way of knowledge transfer from large models into small ones. The aim is to end up with a smaller – and hence faster – yet well-performing model. The steps are 1) to train a big neural classifier model (also called the *teacher*), 2) to let a smaller neural classifier model (the *student*) learn from it – by learning to mimic the teacher's behaviour. Hence also the name *teacher-student knowledge distillation*, often simply *knowledge distillation*.

There are different ways of defining the teacher's "behaviour" which the student learns to mimic. Originally, this was realised as learning to mimic the teacher's predictions: A dataset would be labelled by the teacher, and the student would be trained on these labels (which are in this context referred to as the *hard labels*). The dataset used for training the student (together with the teacher-generated labels) is referred to as the *transfer dataset*.

Later, Ba and Caruana (2014) introduced the idea of learning from the teacher-generated *soft labels*, which are the teacher's logits. The idea is to provide the student with richer information about the teacher's decisions: While hard labels only express which class had the highest predicted probability, soft labels also describe how confident the prediction was and which other classes (and to what extent) the teacher was considering for a given example.

When soft labels were first used, the student's training loss function was the mean squared distance between the student's and the teacher's logits:

$$E_{MSE} = \sum_{c=1}^{C} (z_t^{(c)} - z_s^{(c)})^2 \tag{2.7}$$

where C is the number of classes and z_t, z_s are the teacher's and student's logits. Hinton et al. (2015) proposed a more general approach, addressing the issue of overconfident teachers with very sharp logit distributions. The issue with such distributions is that they carry little additional information beyond the hard label (since the winning class has a huge probability and all others have negligibly small probabilities). To "soften" such sharp distributions, Hinton et al. proposed using the *cross-entropy loss* (2.8) in combination with *softmax with temperature* (2.9) (instead of the standard softmax) in training both the teacher and the student.

$$E_{CE} = \sum_{c=1}^{C} z_t^{(c)} \log z_s^{(c)} \tag{2.8}$$

$$p_c = \frac{\exp\left(z^{(c)}/T\right)}{\sum_{c'=1}^{C} \exp\left(z^{(c')}/T\right)} \tag{2.9}$$

The temperature parameter T determines the extent to which the distribution will be "unsharpened" – two extremes being the completely flat, uniform distribution (for $T \to \infty$) and the maximally sharp distribution[9] (for $T \to 0$). When $T > 1$, the distribution gets softened and the student can extract richer information from it. Today, using soft labels with the cross-entropy loss with temperature is what many refer to simply as knowledge distillation.

Since 2015, further knowledge distillation variants have been proposed, enhancing the vanilla technique in various ways, for example:

- Papamakarios (2015, p. 13) points out that mimicking teacher outputs can be extended to mimicking the *derivatives* of the teacher's loss with respect to the inputs. This is realised by including in the student's loss function also the term: $\frac{\partial \boldsymbol{o}_s}{\partial \boldsymbol{x}} - \frac{\partial \boldsymbol{o}_t}{\partial \boldsymbol{x}}$ ($\boldsymbol{x}$ being an input, e.g. a sentence, and $\boldsymbol{o}$ being the output, e.g. the predicted class).
- Romero et al. (2015) proposed to additionally match the teacher's internal, intermediate representations of the input. Huang and Wang (2017) achieved this by learning to align the distributions of neuron selectivity patterns between the teacher's and the student's hidden layers. Unlike standard knowledge distillation, this approach is no longer limited only to classifier models with softmax outputs (see the approach of Hinton et al. (2015) discussed above).
- Sau and Balasubramanian (2016) showed that learning can be more effective when noise is added to the teacher logits.
- Mirzadeh et al. (2019) showed that when the teacher is much larger than the student, knowledge distillation performs poorly, and improved on this by "multi-stage" distillation: First, knowledge is distilled from the teacher into an intermediate-size "teacher assistant" model, then from the assistant into the final student.

[9] I.e. having the preferred class's probability 1 and the other classes' probabilities 0.

2.3.2 Knowledge distillation in NLP

The knowledge distillation research discussed so far was tied to the image processing domain. This is not surprising: Image processing was the first area to start taking advantage of deep learning, and bigger and bigger models had been researched ever since the revolutional AlexNet (Krizhevsky et al., 2012).

In NLP and in text processing in particular, the (recurrent) models were moderately sized for a long time, not attracting much research in model compression. Still, one early notable work was on adapting knowledge distillation for sequence-to-sequence models (Kim and Rush, 2016), while another pioneering study (Yu et al., 2018) distilled a recurrent model into an even smaller one – to make it suitable for running on mobile devices.

Understandably, the real need for model compression started very recently, when the large pre-trained Transformer models became popular. Large size and low speed seemed to be the only downside of these – otherwise very successful and accessible – models.

Perhaps the first decision to make when distilling large pre-trained models is at which point to distil. In particular, one can distil the general knowledge from a pre-trained teacher and use such a general student by fine-tuning it on downstream tasks, or one can fine-tune the pre-trained teacher on a task and then distil this specialised knowledge into a student model meant for the one task only. Each of these approaches has its advantages and disadvantages.

In the first scenario (distilling pre-trained knowledge), a major advantage is that the distillation happens once and the small student can be fine-tuned quickly for various downstream tasks. Since the distillation can be done on the same data that the teacher was pre-trained on – large unlabelled text corpora –, lack of transfer data is not a concern. A possible risk is that the large amount of general pre-trained language knowledge will not "fit" into the small student, requiring the student itself to be relatively large. Sanh et al. (2019) took this approach and, while their student can be successfully fine-tuned for a wide range of tasks, it is only 40% smaller than the $\text{BERT}_{\text{Base}}$ teacher.

In the second scenario, only the task-specific knowledge needs to be transferred to the student – potentially allowing smaller students. However, teacher fine-tuning and distillation have to be done anew for each task and this is resource-hungry. Additionally, there may be a lack of transfer data if the downstream task dataset is small. Various ways of addressing this issue by *augmenting* small datasets have been proposed, with mixed success. Mukherjee and Awadallah (2019) use additional unlabelled in-domain sentences with labels generated by the teacher – this is limited to cases where such in-domain data are available. Tang et al. (2019b) create additional sentences using simple, rule-based perturbation of existing sentences from the downstream dataset. Finally, Jiao et al. (2019) and Tang et al. (2019a) use large Transformer models generatively to create new sentences. In the first case, BERT is applied repeatedly to an existing sentence, changing words into different ones one by one and thus generating a new sentence. In the second case, new sentences are sampled token-by-token from a GPT-2 model fine-tuned on the downstream dataset with the next-token-prediction objective.

Clearly, each approach is preferred in a different situation: If the requirement is to compress the model as much as possible, and there is enough transfer data, distilling the fine-tuned teacher is more promising. If, on the other hand, one wants to make available a re-usable, small model, then distilling the broader, pre-trained knowledge is preferred.

2.4 Interpreting NLP models

Neural models are by their very nature opaque or even black boxes, and not properly understanding them is a serious concern. Despite the typical preference of performance over transparency, recently, the demand for explainable artificial intelligence (XAI) has been increasing, as neural models become widely used. (Besides the DARPA XAI program[10], conferences like the International Joint Conference on Artificial Intelligence (IJCAI), the SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), and the Conference on Computer Vision and Pattern Recognition (CVPR) now feature dedicated XAI workshops[11].)

The area of image processing has seen the most attempts at interpreting neural models and their behaviour. One reason being that visiual tasks are often doable and easy to reason about for researchers and for humans in general. Various techniques shed light into the behaviour of image classifiers; for instance, techniques for creating images that maximally excite certain neurons (Simonyan et al., 2014), or highlighting those parts of an image that a particular neuron "focuses" on (Zeiler and Fergus, 2014).

In NLP, interpretation is more difficult. Additionally, most research in interpreting NLP models started only relatively recently, after large neural models became widely used. In their review, Belinkov and Glass (2019) note that many methods for analysing and interpreting models are simply adapted from image processing, especially the approach of visualising a single neuron's focus, given an input. In attentional sequence-to-sequence models, the attention maps can be visualised to explore the soft alignments between input and output words (see, e.g., Strobelt et al. (2019)). However, these methods are mostly qualitative and suitable for exploring individual input examples, thus not well suited for drawing statistically backed conclusions or for quantitative model comparison.

More quantitative and NLP-specific are the approaches that explore the linguistic knowledge present in a model's internal representations. Most often, this is realised by *probing* the representations for specific linguistic knowledge: trying to automatically recover from them specific properties of the input. When such recovery works well, the representations must have contained the linguistic knowledge tied to the input property in question. First used by Shi et al. (2016) for exploring syntactic knowledge captured by machine translation models, this general approach was quickly adopted more widely. Adi et al. (2017) explored sentence encodings from recurrent models by probing for simple properties like sentence length, word content and word order. More recently, Conneau et al. (2018) curated a set of 10 probing tasks ranging from easy surface properties (e.g. sentence length) through syntactic (e.g. the depth of the syntactic parse tree) to semantic ones (e.g. identifying semantically disrupted sentences). Focusing on Transformers, Tenney et al. (2019b) proposed a set of *edge probing* tasks, examining how much contextual knowledge about an entire input sentence is captured within the contextual representation of one of its

[10] www.darpa.mil/program/explainable-artificial-intelligence

[11] See sites.google.com/view/xai2019/home, xai.kdd2019.a.intuit.com/, explainai.net/.

words. Their tasks correspond to the typical steps of a text processing pipeline – from part-of-speech (POS) tagging to identifying dependencies and entities to semantic role labelling. Tenney et al. (2019a) managed to localise the layers of BERT most important for each of these "skills". They showed that the ordering of these "centres of expertise" within BERT's encoder matches the usual low- to high-level order: from simple POS tagging in the earlier layers to more complex semantic tasks in the last layers.

While the discussed approaches provide valuable insights, they merely help us intuitively describe or quantify the kinds of internal knowledge/expertise present in the models. Gilpin et al. (2018) call this level of model understanding *interpretability* – comprehending what a model does. However, they argue that what we should strive to achieve is *explainability*: the ability to "summarize the reasons for neural network behavior, gain the trust of users, or produce insights about the causes of their decisions". In this sense, today's methods achieve only interpretability because they enable researchers to describe but not explain – especially in terms of causality – the internals and decisions of the models. Still, interpreting models is an important step not only towards explaining them, but also towards understanding the properties of different architectures and methods and improving them.

2.5 Summary

Since around 2013, the area of NLP has been taking advantage of deep neural models. With the introduction of Transformers, the models became even deeper and more powerful. Today's pre-trained Transformer-based models like BERT make state-of-the-art NLP relatively accessible, but the models are often too large and slow for practical applications. Compressing such models has become an active research area, with knowledge distillation being a particularly successful compression technique. However, the self-attentional, Transformer-based models, as well as compressing them, are still relatively young concepts. More research is needed to better interpret the behaviour of models like BERT, and to better understand the nature of the knowledge transfer from large Transformers into smaller, compressed ones.

Chapter 3

Datasets

In this chapter, I introduce the different datasets used throughout the work:

1. To later experiment with models in the context of a wide range of NLP tasks, I use different small **downstream task datasets** on which I train large Transformer models.
2. For knowledge distillation from the large into smaller models, large **transfer datasets** are used, created from the downstream datasets using data augmentation.
3. Finally, **probing datasets** are used for analysing the linguistic capabilities of the large and the small models.

3.1 Downstream tasks

The downstream task datasets I use to fine-tune the teacher model. The tasks are chosen to be diverse so that the knowledge distillation analysis later in this work is set in a wide NLP context. At the same time, all the datasets are rather small and therefore well representing the type of use case where pre-trained models like BERT are desirable due to the lack of labelled fine-tuning data.

Today, perhaps the most widely used collection of challenging NLP tasks[1] is the GLUE benchmarking collection (Wang et al., 2018). This collection comprises 11 tasks which enable model benchmarking on a wide range of NLP problems from sentiment analysis to detecting textual similarity, all framed as single-sentence or sentence-pair classification. Each task comes with an official scoring metric (such as accuracy or F1), labelled training and evaluation datasets, and a testing dataset with labels not released publicly. The test-set score accumulated over all 11 tasks forms the basis for the popular GLUE leaderboard[2].

In this work, I use single-sentence classification tasks (i.e. not sentence-pair tasks). Therefore, only two GLUE tasks are suitable for my purposes – the Corpus of Linguistic Acceptability (CoLA) and the Stanford Sentiment Treebank in its binary classification variant

[1] Challenging by the nature of the tasks and by the small dataset size.
[2] gluebenchmark.com/leaderboard

(SST-2). Additionally, I choose a third task to make my work cover the area of conversational language. This way, I build on my previous research in compressing BERT for conversational tasks (Sucik, 2019), undertaken as part of an internship with Rasa[3], a company building open-source tools for conversational AI. The third dataset, called Sara, focuses on classifying human messages (from human-bot conversations) according to their intent.

3.1.1 Corpus of Linguistic Acceptability

The CoLA dataset (Warstadt et al., 2019) comprises roughly 8,500 training sentences, 1,000 evaluation and 1,000 testing sentences. The task is to predict whether a given sentence represents acceptable English or not (binary classification). All the sentences are collected from linguistic literature where they were originally hand-crafted to demonstrate various linguistic principles and their violations.

The enormous variety of principles, together with many hand-crafted sentences that comply with or violate a principle in a niche way, make this dataset very challenging even for the state-of-the-art Transformer models. As a non-native speaker, I myself struggle with some of the sentences, for instance:

- **The car honked down the road.* (unacceptable[4])
- *Us, we'll go together.* (acceptable)

There are many examples which are easy for humans to classify but may be challenging for models which have imperfect understanding of the real world. Sentences like "Mary revealed himself to John." require the model to understand that "Mary", being a typical female name, disagrees with the masculine "himself".

The scoring metric is Matthew's Correlation Coefficient (MCC) (Matthews, 1975), a correlation measure between two binary classifications. The coefficient is also designed to be robust against class imbalance, which is important because the dataset contains many more acceptable examples than unacceptable ones[5].

3.1.2 Stanford Sentiment Treebank

The SST-2 dataset (Socher et al., 2013) is considerably bigger than CoLA, with roughly 67,000 training examples, 900 evaluation and 1,800 testing examples. It contains sentences and phrases from movie reviews collected on `rottentomatoes.com`. The main SST dataset comes with human-created sentiment annotations on the continuous scale from very negative to very positive. SST-2 is a simplified version with neutral-sentiment phrases removed, only containing binary sentiment labels (positive and negative).

Unlike the hand-crafted examples in CoLA, many examples in SST-2 are not the best-quality examples. In particular, sentences are sometimes split into somewhat arbitrary

[3]`rasa.com`

[4]The "*" is a standard way to mark ungrammatical sentences in linguistic literature.

[5]For details on the class imbalance, see Fig. A.1 in Appendix A.

segments[6], such as:

- *should have been someone else -* (negative)
- *but it could have been worse.* (negative)

The labels are also sometimes unclear, see:

- *american chai encourages rueful laughter at stereotypes only an indian-american would recognize.* (negative)
- *you won't like roger, but you will quickly recognize him.* (negative)

Despite the problematic examples, most are straightforward (e.g. "delightfully cheeky" or "with little logic or continuity"), making this task a relatively easy one. With accuracy being the official metric, best models in the GLUE leaderboard score over 97%, very close to the official human baseline of 97.8%[7].

3.1.3 Sara

As the third task, I use an intent classification dataset created by Rasa, a start-up building open-source tools for conversational AI[8].

The dataset is named Sara after the chatbot deployed on the company's website[9]. The Sara chatbot is aimed for holding conversations with the website visitors on various topics, primarily answering common questions about Rasa and the tools that it develops (the same tools were used to build Sara). Simultaneously, the Sara dataset is used for most of research at Rasa. To support diverse topics, Sara internally classifies each human message as one of 57 intents and then generates an appropriate response. The Sara dataset is a collection of human-generated message examples, each manually labelled with one of the 57 intents, e.g.:

- *what's the weather like where you are?* (ask_weather)
- *what is rasa actually* (ask_whatisrasa)
- *yes please!* (affirm)
- *i need help setting up* (install_rasa)
- *where is mexico?* (out_of_scope)

For a list of all intents, explained and accompanied with real examples from the dataset, see Tab. A.1 in Appendix A.

In the early days of the chatbot, it supported fewer intents, and several artificial examples per intent were first hand-crafted by Rasa employees to train the initial version of Sara's intent classifier. After Sara was deployed, more examples were collected and annotated

[6]This is due to the use of an automated parser in creating the dataset.

[7]See the GLUE leaderboard at `gluebenchmark.com/leaderboard`

[8]For transparency: My co-supervisor for this work – Vladimir Vlasov – is a Rasa employee, and he also supervised me during my Machine learning research internship with Rasa in the summer of 2019.

[9]See the bot in action at `rasa.com/docs/getting-started/`.

from conversations with the website's visitors[10]. Inspired by the topics that people tended to ask about, new intent categories were added. Today, the dataset still evolves and can be found – together with the implementation of Sara – at github.com/RasaHQ/rasa-demo. It contains both the original hand-crafted examples as well as the (much more abundant) examples from real conversations.

The Sara dataset version I use dates back to October 2019, when I obtained it from Rasa and pseudonymised the data[11]. In particular, I removed any names of persons and e-mail addresses in any of the examples, replacing them with the special tokens `__PERSON_NAME__` and `__EMAIL_ADDRESS__`, respectively. The dataset comprises roughly 4,800 examples overall, and was originally split into 1,000 testing examples and 3,800 training examples. I further split the training partition into training and evaluation, with roughly 2,800 and 1,000 examples, respectively. All three partitions have the same class distribution.

In line with how the dataset is used for research at Rasa, I use as the main scoring metric the *multi-class micro-averaged F1 score* ($\mathrm{F1}_{micro}$), even though other reasonable metrics exist. First of all, in the binary classification case, the F1 score balances two desirable properties of any classifier: precision P and recall R: $\mathrm{F1} = \frac{2PR}{P+R}$. P quantifies the purity of reported positives: $P = TP/(TP+FP)$, R quantifies the reported portion of all positives: $R = TP/(TP+FN)$ (where TP are true positives, FP are false positives, and FN are false negatives). In classification with more than 2 classes, one can still compute the F1 score with respect to each individual class (treating the multi-class classification as a collection of binary classification decisions). Taking the average of such class-specific F1 scores leads to the *macro-averaged* F1 metric:

$$\mathrm{F1}_{macro} = \frac{1}{C}\sum_c \mathrm{F1}_c = \frac{1}{C}\sum_c \frac{2P_cR_c}{P_c+R_c}, \quad C \text{ being the number of classes} \tag{3.1}$$

While this metric quantifies the F1 score on an "average" class, it does not account for different class sizes. In particular, if there are many small classes with little data to learn from and hence with low F1 scores, then the average F1 will be pulled down – even if the classifier succeeds on most data, which belongs to several big classes. One way to deal with these undesirable effects of class imbalance is to use the *micro-averaged* F1 score. As its name suggests, it can be thought of as F1 averaged not on the macro level (classes), but on the micro level (individual examples), where the F1 score for a single example is 1 for a correct prediction (this follows from the standard formula $\mathrm{F1} = \frac{2PR}{P+R}$) and 0 for an incorrect prediction (by definition):

$$\mathrm{F1}_{micro} = \frac{1}{N}\sum_n \mathrm{F1}_n = \frac{1}{N}\sum_n \begin{cases} \text{if correct} & 1 \\ \text{else} & 0 \end{cases}, \quad N \text{ being the number of examples} \tag{3.2}$$

This score does take into account class imbalance because each example has "one vote" in the averaging process. Therefore, it is well suited for situations where the classifier

[10] To get consent for such use of the conversations, each visitor was shown the following before starting a conversation with Sara: "Hi, I'm Sara! By chatting to me you agree to our privacy policy.", with a link to rasa.com/privacy-policy/

[11] As a former employee of Rasa, I got access to the data under the NDA I had signed with the company. I had permission from Rasa to use the pseudonymised data for this project; the use complied with the ethical approval process of Rasa.

should perform well on many examples, not necessarily on many classes (as there can be many classes that are insignificant). Additionally, the $F1_{micro}$ score has the same value as accuracy.

3.2 Data augmentation for larger transfer datasets

As discussed in Sec. 2.3.2, knowledge distillation works best with large amounts of data used as the transfer datasets. When the transfer dataset is small, it does not provide enough opportunity for the teacher to "demonstrate its knowledge" to the student, and the student learns little. Therefore, for each downstream task, I create a large transfer dataset by "inflating" the small training portion of the corresponding downstream dataset – by augmenting it with additional sentences. I then add teacher logits to such augmented dataset, and use it to train the student models.

Tang et al. (2019a) demonstrated on several GLUE tasks that using an augmented training portion for distillation leads to much better student performance than using just the original small training portion. For CoLA in particular, using just the small original training set led to very poor student performance (see Table 1 in Tang et al.).

I take the augmentation approach that Tang et al. found to work the best: Generating additional sentences using a GPT-2 model (Radford et al., 2019) fine-tuned on the training set[12]. The steps for creating the transfer dataset from the training portion are:

1. Fine-tune the pre-trained GPT-2 model (the 345-million-parameter version) on the training portion for 1 epoch (where an epoch is one complete pass through all training examples) with the language-modelling objective (i.e. predicting the next subword token given the sequence of tokens so far).

2. Sample from the model a large number of tokens to be used as the beginnings (*prefixes*) of the augmentation sentences. This sampling can be done as one-step next-token prediction given the special `SOS` (start-of-sentence) token.

3. Starting from each sampled prefix, generate an entire sentence token by token by repeatedly predicting the next token using the GPT-2 model. The generation of a sentence stops when the special `EOS` (end-of-sentence) token is generated or when the desired maximum sequence length is reached – in this case 128 tokens.

4. Add the generated augmentation sentences to the original training data, and generate the teacher logits for each sentence.

For consistency with Tang et al. (2019a), I added 800,000 augmentation sentences to the training data of each of the three downstream tasks, resulting in the transfer datasets comprising roughly 808,500, 867,000, and 802,800 sentences for CoLA, SST-2, and Sara, respectively.

[12] I used the code for Tang et al. (2019a) which is available at `github.com/castorini/d-bert`.

3.3 Probing tasks

The probing tasks (discussed in Sec. 2.4) I use after knowledge distillation to analyse the linguistic capabilities of the students and the teacher. In particular, I use the probing suite curated by Conneau et al. (2018), consisting of 10 tasks[13].

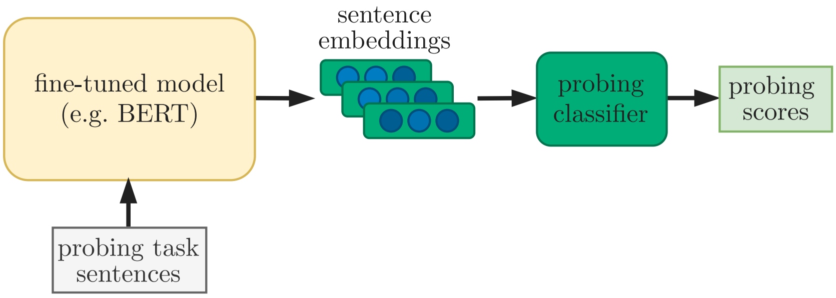


Figure 3.1: A high-level diagram of the probing process.

Each probing task is a collection of 120,000 labelled sentences, split into training (100,000), evaluation (10,000) and test (10,000) set. The label refers to a property of the sentence, such as the sentence's length. The aim is to recover the property from an encoding of the sentence, produced by the model being probed. Fig. 3.1 shows the basic workflow. First, the model is used to produce an encoding of each sentence. Then, a light-weight classifier is trained, taking the training sentences' encodings as inputs and learning to produce the labels. The evaluation sentence encodings are used to optimise the hyper-parameters of the classifier. Finally, a probing score (accuracy) is produced on the test encodings. The score quantifies how well the sentence property in question is recoverable (and thus present) in the encodings. This serves as a proxy measure of the linguistic knowledge tied to the property. If, for instance, the property to be recovered is the depth of a sentence's syntactic parse tree, the score hints at the model's (un)capability to understand (and parse) the syntax of input sentences. By extracting probing encodings from different parts of a model (e.g. from different layers), the probing scores can additionally serve as cues for localising the linguistic knowledge in question – one can observe how the amount of this knowledge varies across different model parts and where it is most concentrated.

Regarding the linguistic capabilities explored by the probing suite, each task falls into one of three broad categories – surface properties, syntax, and semantics:

1. Surface information:
    - **Length** is about recovering the length of the sentence. The labels are somewhat simplified: The actual sentence lengths grouped into 6 equal-width bins – making this task a 6-way classification.
    - **WordContent** is about identifying which words are present in the sentence. A collection of 1000 mid-frequency words was curated, and sentences were

[13] The data, along with code for probing neural models, are publicly available as part of the SentEval toolkit for evaluating sentence representations (Conneau and Kiela, 2018) at `github.com/facebookresearch/SentEval`.

chosen such that each contains exactly one of these words. The task is to identify which one (1000-way classification).

2. Syntactic information:

   - **Depth** is about classifying sentences by their syntactic parse tree depth, with depths ranging from 5 to 12 (hence 8-way classification).

   - **BigramShift** is about sensitivity to (un)natural word order – identifying sentences in which the order of two randomly chosen adjacent words has been swapped (binary classification). While syntactic cues may be sufficient to identify an unnatural word order, intuitively, broken semantics can be another useful signal – thus making this task both syntactic and semantic.

   - **TopConstituents** is about recognising the top syntactic constituents – the nodes found in the syntactic parse tree just below the S (sentence) node. This is framed as 20-way classification, choosing from 19 most common top-constituent groups + the option of "other".

3. Semantic information:

   - **Tense** is a binary classification task, identifying the tense (present or past) of the sentence's main verb (the verb in the main clause). At the first sight, this is mainly a morphological task (in English, most verbs have the past tense marked by the "-d/ed" suffix). However, the model first has to identify the main verb within a sentence, which makes this task also syntactic and semantic.

   - **SubjNumber** is about determining the number (singular or plural) of the sentence's subject (binary classification). Similar to the previous task, this one (and the next one too) is arguably about both morphology and syntax/semantics.

   - **ObjNumber** is the same as SubjNumber, applied to the direct object of a sentence.

   - **OddManOut** is binary classification, identifying sentences in which a randomly chosen verb or noun has been replaced with a different random verb or noun. Presumably, the random replacement in most cases makes the sentence semantically unusual or invalid (e.g. in "He reached inside his persona and pulled out a slim, rectangular black case." the word "persona" is clearly odd). To make this task more difficult, the replacement word is chosen such that the frequency of the bigrams in the sentence stays roughly the same. (Otherwise, in many cases, the random replacement would create easy hints for the probing classifier, in the form of bigrams that are very unusual.)

   - **CoordinationInversion** works with sentences that contain two coordinate clauses (typically joined by a conjunction), e.g. "I ran to my dad, but he was gone." In half of the sentences, the order of the two clauses was swapped, producing sentences like: "He was gone, but I ran to my dad." The task is to identify the changed sentences (which are often semantically broken).

When choosing from the existing probing suites, I considered that of Tenney et al. (2019a) as well. As the authors showed, their tasks and methods can effectively localise different types of linguistic knowledge in a Transformer model like BERT. However, the task data are not freely available, the tasks have a relatively narrow coverage with a heavy focus on the most complex NLP tasks like entity recognition and natural language inference, and the probing is done on single-token representations. The suite of Conneau et al., on the other hand, is publicly available, better covers the easier tasks (surface and syntactic information), and examines whole-sentence representations. One interesting direction for future work is to use both of these probing suites, compare the results they lead to (in particular their agreement), and explore the extent to which the different probing approaches complement each other.

3.4 Summary

I have introduced the three different types of data used in this work. These types also define the skeleton of my experiments and analyses:

1. First, I train one teacher model for each of the three downstream datasets.
2. Then, each teacher teaches two students, using the transfer dataset as the "carrier" of the teacher knowledge.
3. Finally, the linguistic skills of the students as well as the teachers are measured and analysed using the probing tasks.
4. Additionally, the downstream task sentences are used for analysing the prediction characteristics of each model.

While using the GLUE benchmark tasks is the usual way of comparing and analysing sentence encoder models, none of the tasks focuses on the conversational domain. I use an additional downstream task – Sara – to make this work more relevant for the area of conversational AI. My prior familiarity with the Sara dataset can be an advantage when later analysing the individual predictions of the teacher and student models on the downstream datasets.

Chapter 4

Methods and Implementation

This chapter elaborates on the main objectives of this work, the knowledge distillation and model analysis approaches I took, and goes into detail in describing the design and implementation work underlying my experiments.

4.1 Methods and objectives

The main aim is to explore the use of knowledge distillation. In particular, it is used on three different NLP tasks (CoLA, SST-2, Sara) and with two different student architectures: a bidirectional LSTM student and a BERT student. An analysis stage follows, where I look at and compare the teacher and students on each task. Note that the focus is not on improving scores reported by previous works, or on finding the best hyperparameter configurations; I aim to learn more about knowledge distillation.

Being inspired by my internship at Rasa on compressing BERT[1], this work aims to produce student models as small as possible. Therefore, I take the approach of first fine-tuning a teacher model and then distilling the fine-tuned knowledge into small students (for the other option, refer back to the discussion in Sec. 2.3.2).

Creating small yet well-performing students requires not just setting up an implementation of knowledge distillation, but also optimising the student models' hyperparameters. Even if extensive optimisation is not the main goal, the models used for further analysis should reach reasonable performance levels in order for the analysis to be of real value. However, in order to constrain the amount of optimisation, I carry out a relatively thorough hyperparameter exploration only on the CoLA task. Subsequently, the best parameters are applied on the other two tasks, with only the most essential decisions – like the student model size – made on each task separately.

The analysis stage of this work inspects what the two students learnt well and what they did not, how they differ from their teacher and from each other. Where possible, I try to produce conclusions that generalise across the three downstream datasets.

[1] See `blog.rasa.com/compressing-bert-for-faster-prediction-2/` and `blog.rasa.com/pruning-bert-to-accelerate-inference/`.

As the first analysis approach, all models are probed for various types of linguistic knowledge. This produces simple, quantitative results, which, however, are not necessarily easy to interpret.

As the second approach, I carry out a – mostly qualitative – analysis of the models' predictions on concrete sentences. This approach is not widely reported, despite being simple in nature – manually inspecting a model's predictions on a case-by-case basis follows from the natural curiosity of an empirical scientist. While it involves a lot of human labour and does not guarantee easy-to-interpret, quantitative results, I still make use of this approach and try to gain qualitative insights. In particular, the predictions are inspected both in terms of correctness – e.g. manually analysing sentences which were classified correctly by one model but not by another – and through confidence – which models are more confident, on what sentences are they (un)confident, and how this relates to their (in)correctness.

Finally, the results of probing and prediction analysis are juxtaposed. I ask whether the two approaches agree or disagree, and whether they shed light on the same or different aspects of the models and of knowledge distillation.

Because of the unavailability of test-set labels in CoLA and SST-2, the prediction analysis is carried out on the evaluation set for each downstream task. This can be understood as inspecting the model qualities being optimised when one tunes a model's hyperparameters on the evaluation data. Another option would be to carry out the analysis on a held-out set not used in training.

4.2 System overview and adapted implementations

Because a lot of research around Transformers is open-sourced, my work makes use of multiple existing codebases. Fig. 4.1 shows the high-level pipeline of this project. It is inspired by the best pipeline of Tang et al. (2019a), although they only used the BiLSTM student and did not carry out probing or prediction analysis.

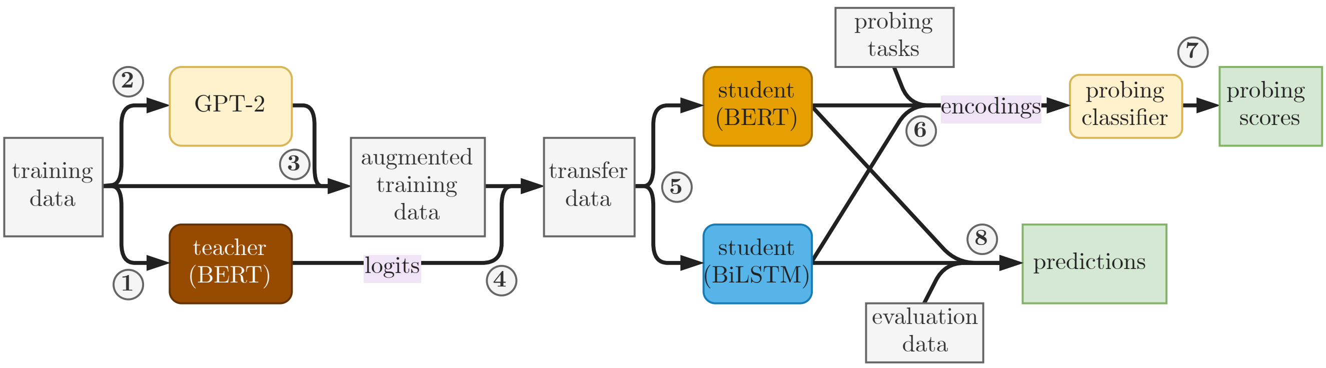


Figure 4.1: The main pipeline of this work: 1 teacher fine-tuning, 2 GPT-2 fine-tuning, 3 generating augmentation sentences, 4 adding teacher logits to the augmented training dataset, 5 knowledge distillation into students, 6 producing probing sentence encodings, 7 training the probing classifier and producing probing scores, 8 producing predictions on evaluation sentences.

For most of the implementation, the `transformers` open-source PyTorch library (Wolf et al., 2019)[2], is used, which provides tools for working with pre-trained Transformers like BERT. For knowledge distillation, I adapt the code of Sanh et al. (2019), which is today also part of `transformers`[3]. (Note that the authors apply knowledge distillation *before* downstream fine-tuning.) For augmenting the training data using GPT-2 and for knowledge distillation with the BiLSTM student, I adapt the code of Tang et al.[4], which uses an early version of `transformers`. For probing the two students, the SentEval framework (Conneau and Kiela, 2018)[5] is used.

My own contributions to the implementation lie primarily in adapting and integrating the different codebases into one, and in adding the possibility for optimising a range of student hyperparameters. I also make the code more flexible, relative to the original codebases which encode numerous fixed design decisions made by Sanh et al. and Tang et al.. The core of my implementation is open-sourced as a fork of the `transformers` library at `github.com/samsucik/pytorch-transformers/`, while the implementation needed for individual experiments, analyses, and reporting, resides at `github.com/samsucik/knowledge-distil-bert`.

4.3 Implementation details

4.3.1 Teacher fine-tuning

Following Tang et al. (2019a), the case-insensitive pre-trained $\text{BERT}_{\text{Large}}$ is used as the teacher model (from now, referred to as BERT_{T}). With $L = 24$ encoder layers, $A = 16$ self-attention heads, the hidden dimension $d_h = 1024$ and the intermediate dimension $d_I = 4096$, the model has 340 million trainable parameters (as discussed in more detail previously in Sec. 2.2.2). The large BERT variant generally performs better than the 110-million-parameter $\text{BERT}_{\text{Base}}$ variant (both variants published by Devlin et al. (2019)) and is therefore more attractive, but also slower, with a greater incentive for compression.

loss function	cross-entropy
learning algorithm	Adam ($\eta = 5 \times 10^{-5}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$)
training budget	3 epochs
η scheduling	linear warm-up (first 10% of training), then linear decay (see Fig. 4.2)
batch size	36
dropout rate	0.1

Table 4.1: The fine-tuning configuration of the teacher BERT model.

For teacher fine-tuning on each downstream task, the procedure of Tang et al. is used, summarised in Tab. 4.1. While the performance of BERT_{T} converges (flattens) within

[2] `github.com/huggingface/transformers`
[3] `github.com/huggingface/transformers/tree/master/examples/distillation`
[4] `github.com/castorini/d-bert`
[5] `github.com/facebookresearch/SentEval`

the 3-epoch training budget on CoLA and SST-2, the convergence is much slower for Sara. Hence, I empirically found a more suitable number of epochs within which the teacher converges on Sara: 10. See Fig. 4.2 for the evaluation-set performance of the teacher models and how they converge during fine-tuning.

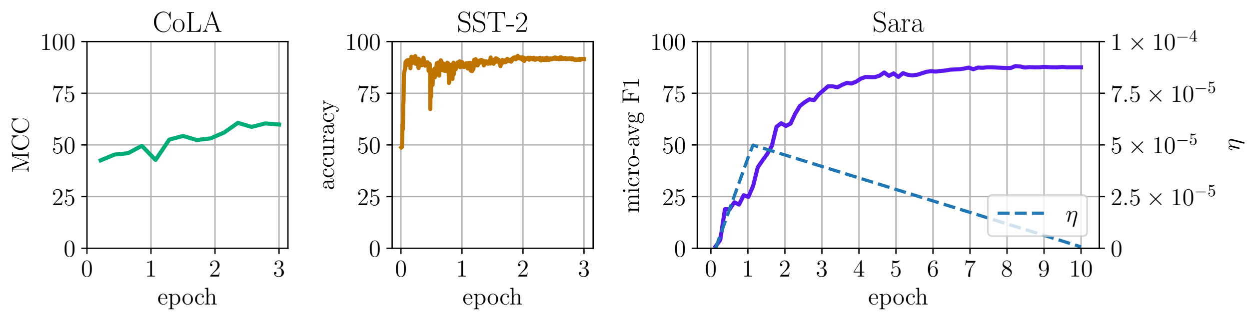


Figure 4.2: The evaluation-set performance of teacher models across fine-tuning, together with an illustration of the learning rate scheduling for $\mathrm{BERT_T}$ on Sara. Unintentionally, I used different logging frequencies in fine-tuning the teachers, hence the SST-2 plot is dense (and appears more noisy) while the CoLA plot is sparse.

4.3.2 Augmentation with GPT-2

In fine-tuning the GPT-2 model, again the procedure of Tang et al. is used (summarised in Tab. 4.2). This is very similar to the fine-tuning configuration used for $\mathrm{BERT_T}$, with small differences. The AdamW learning algorithm is used (Loshchilov and Hutter, 2019), which is a variant of Adam with weight decay imposed on all learnable parameters, making the values slowly decay towards 0 in the absence of learning. The decay rate λ determines the fraction by which each weight decays at each training step. The only parameter I choose differently from Tang et al. is the batch size B: While they use batches of 48 examples, I only process examples in batches of 16, in order to make the fine-tuning possible with the limited memory resources.

loss function	cross-entropy
learning algorithm	AdamW ($\eta = 5 \times 10^{-5}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\lambda = 1 \times 10^{-3}$)
training budget	1 epoch
η scheduling	linear warm-up (first 10% of training), then linear decay
batch size	16
dropout rate	0.1

Table 4.2: The fine-tuning configuration of the GPT-2 model.

4.3.3 BiLSTM student model

As the first student, I use the bidirectional LSTM (BiLSTM) from Tang et al. (see Fig. 4.3). The model comprises in particular one hidden BiLSTM layer with 300 units, which is composed of two LSTM layers processing the inputs in opposite directions. The last hidden states for either of the two processing directions are concatenated and passed to a fully connected layer with 400 output units[6], which uses the rectified linear unit (ReLU) activation function (Nair and Hinton, 2010), and dropout. A final (linear) layer follows, projecting to the number of target classes, i.e. producing the logits. The model is topped with a softmax classifier for normalising the logits and producing class probabilities.

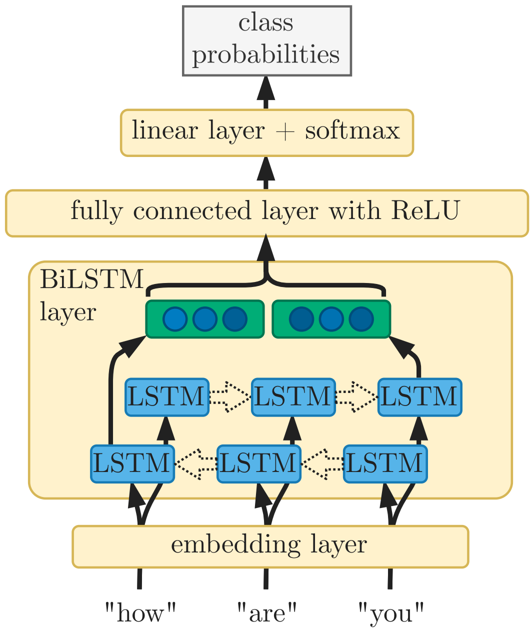


Figure 4.3: The bidirectional LSTM student. Diagram adapted from Figure 1 of Tang et al. (2019b).

The original model was built to process sentences word by word, encoding each word using the pre-trained word2vec embeddings[7] before passing it to the LSTM layer. Words for which there is no embedding (*out-of-vocabulary* words, or just OOV) are embedded using a vector initialised with random numbers drawn uniformly from [-0.25, 0.25]. The embedding layer supports three *embedding modes*, based on Kim (2014):

1. **Static**: the embedding parameters are frozen and do not change during training.
2. **Non-static**: the embedding parameters are allowed to change (fine-tune) during training.
3. **Multichannel**: two embedding instances are used in parallel, one is frozen, the other one is allowed to change. For each input word, the two embeddings produced are

[6] Even though Tang et al. tried also other, slightly different layer dimensions, these are the ones that worked the best on CoLA.

[7] The 300-dimensional version trained on Google News, see `code.google.com/archive/p/word2vec/`.

concatenated together for further processing. The multichannel mode is the one used by Tang et al..

One significant change I made to this model is enabling the use of wordpiece embeddings instead of word-level ones. This way, the fine-tuned embedding parameters from BERT_T can be used to initialise the student's embedding layer, providing some of the teacher's "knowledge" even before the student training (knowledge distillation) begins.

When the word2vec embeddings are used, the embedding matrix of the LSTM is constructed in the following way:

1. A vocabulary of all distinct words present in the transfer dataset is made.
2. Only the word2vec vectors corresponding to these words are taken and put together to create the embedding matrix.

This way, even though the full word2vec collection covers 3,000,000 words, the word-level embedding matrix (whether used by the LSTM student or the BERT student) has fewer entries. For the particular transfer datasets I use, the vocabulary has 243,120 words for CoLA, 284,273 words for SST-2, and 172,183 words for Sara.

In total, this model – from now referred to as LSTM_S – has 2.41 million trainable parameters (excluding the embedding layer – this is how model sizes are reported everywhere further in this work[8]), making it 140x smaller than BERT_T.

4.3.4 BERT student model

For the second student, a down-scaled version of BERT_Large is used, matched for size with LSTM_S. In particular, I scale all the dimensions of BERT_Large down by a factor of ~5, leading to a smaller BERT with $L = 5$ encoder layers, the hidden dimension $d_h = 204$, the intermediate dimension $d_I = 750$, and $A = 3$ self-attentional heads – amounting to 2.42 million trainable parameters (embedding parameters excluded). This model is from now referred to as BERT_S.

4.3.5 Knowledge distillation

While Tab. 4.3 summarises the initial configuration of both student models, I elaborate more on these parameters in the rest of this section.

During knowledge distillation, BERT_S is trained using the cross-entropy loss. The softmax temperature is fixed at $T = 3$.[9]

[8] In line with how the number of model parameters is reported by others, for instance in Tang et al. (2019a,b), but also in the GLUE benchmark leaderbord. One reason for this is that the number of embedding parameters is not directly related to the model size and mostly depends on the type of embeddings used – the embedding vocabulary size and the dimensionality of each embedding.

[9] Usual values are from 1 (no effect) to 3. For instance, Sanh et al. (2019) use $T = 2$. In a work that is much closer to my situation, Tsai et al. (2019) apply knowledge distillation from BERT_Base into a 18-million-parameter smaller BERT, observing that from $T = \{1,2,3\}$ the best one was $T = 3$.

Originally, both students were implemented to use random initialisation from scratch before training, with the exception of the embedding layer of LSTM$_\mathrm{S}$, which was initalised from word2vec. Later, I explore different ways of initialising the embedding layers.

LSTM$_\mathrm{S}$ uses the mean squared error (MSE) loss, following Tang et al. who report that MSE led to slightly better performance (compared to cross-entropy loss with $T = 3$). Following preliminary experiments on CoLA, I set the training budget to 30 epochs for LSTM$_\mathrm{S}$ (same as Tang et al.). BERT$_\mathrm{S}$ converges slower and therefore uses a 60-epoch training budget in all following experiments. In student training, the evaluation-set performance reported is always for the best model checkpoint as observed during training; in particular, it may not be the final model version. Using this approach, even if a student's performance eventually starts to decrease during training, the best-performing version is retained for further analysis and comparison.

Following Tang et al., the Adadelta learning algorithm (Zeiler, 2012) with $\eta = 1.0$ and $\rho = 0.95$ is used for training LSTM$_\mathrm{S}$, while Adam is used for BERT$_\mathrm{S}$. Note that Adam is an improved successor of Adadelta[10] and is much more widely used; later in this work, I explore the use of Adam for LSTM$_\mathrm{S}$. No η scheduling is used with LSTM$_\mathrm{S}$, while BERT$_\mathrm{S}$ uses scheduling similar to BERT$_\mathrm{T}$. To prevent "gradient explosion" in LSTM$_\mathrm{S}$, the total magnitude of all gradients is clipped to 30.0 before every parameter update (as used by Tang et al.); however, throughout all experiments, I never observe the gradient norm to reach this limit. For more robust training, the standard dropout rate of 0.1 is used during training of both students, following Devlin et al. (2019) and Tang et al.. Tang et al. report the small batch size $B = 50$ to work well with the BiLSTM student. For BERT$_\mathrm{S}$, I initially use a larger batch size $B = 256$.

While LSTMs can process sequences of any lengths, Transformer models like BERT impose a maximum sequence length for practical reasons, with all sequences within a batch padded to the maximum length. Although BERT$_\mathrm{T}$ allows sequences of up to 512 wordpieces in length, extremely few sentences reach this length – especially in this work, where all inputs are single sentences, not sentence pairs. Therefore, to accelerate training, I use the maximum sentence length of 128 tokens for BERT$_\mathrm{S}$.

	LSTM$_\mathrm{S}$	BERT$_\mathrm{S}$
loss function	mean square error	cross-entropy, $T = 3$
learning algorithm	Adadelta ($\eta = 1.0$, $\rho = 0.95$)	Adam ($\eta = 5 \times 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.98$)
training budget	30 epochs	60 epochs
η scheduling	none	linear warm-up (10 epochs), then linear decay
batch size	50	256
embedding layer initialisation	word2vec	wordpiece, from BERT$_\mathrm{T}$

Table 4.3: The initial parameters of both student models.

[10] In particular, the adaptive mechanism of Adadelta considers only the recent squared gradient magnitudes, whereas Adam also considers the simple (not squared) gradients.

4.3.6 Probing

For the light-weight probing classifier, Conneau et al. (2018) use a small neural network comprising one hidden layer with the sigmoid activation function and dropout, followed by a linear layer projecting to the desired number of classes. In training the classifier, early stopping is used, i.e. training stops when the evaluation-set accuracy does not improve over 5 consecutive iterations. For consistency with the exact method of Conneau et al., I tune the dropout rate (choosing from [0.0, 0.1, 0.2]) and the hidden layer width (choosing from [50, 100, 200]) using the evaluation set. Each probing score is reported as the one for the best dropout and layer width values. Tab. 4.4 summarises all important training parameters.

loss function	cross-entropy
learning algorithm	Adam ($\eta = 1 \times 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$)
training budget	4 epochs (early stopping)
batch size	64
dropout rate	[0.0, 0.1, 0.2]

Table 4.4: The training configuration of the probing classifier.

When probing a model, an important design decision is how to extract sentence representations from the model's layers. The BiLSTM layer of LSTM_S can produce at each timestep two hidden states (one for each processing direction). Conneau et al. experiment with:

1. Creating a *BiLSTM-max* encoding such that each of its elements is the maximum over the values for each timestep. (The encoding has the same dimensionality as the BiLSTM layer output.)

2. Creating a *BiLSTM-last* encoding by simply taking the last hidden state in each direction – the encoding is the same as the BiLSTM layer output.

Conneau et al. report mixed results, with BiLSTM-max encodings leading to better probing scores on some of the probing tasks. I am constrained to using BiLSTM-last since the PyTorch implementation of LSTMs does not give access to intermediate hidden states, only to the last one in each direction.

In BERT, all hidden representations produced by each encoder layer can be accessed. I try three different ways of combining a sequence of hidden representations from a particular layer into a single encoding:

1. Maximum pooling, equivalent to BiLSTM-max: Taking the maximum value for each element over all hidden representations.

2. Single-position encoding (the equivalent of BiLSTM-last): Taking the hidden representation that is used for the final classification. While in LSTM_S, this would mean taking the last hidden state in each direction, in BERT, it is the hidden representation of the first token (the special `[CLS]` token).

3. Average pooling (not explored by Conneau et al.): Similarly to maximum pooling, this uses the average of each element across all representations.

After conducting simple preliminary probing experiments with BERT_T on each downstream task, I observed that the differences between the three approaches are mostly inconsistent and small. However, in many cases, maximum pooling produced worse probing scores than the other two techniques, and average pooling slightly outperformed single-position representations. In all further probing experiments with BERT_T and BERT_S, the average pooling approach is used.

Inspired by the localisation experiments of Tenney et al. (2019a), I probe various encoder layers across the BERT models in order to also localise each type of language knowledge within the model's architecture.

4.4 Computing environment and runtimes

All major parts of my experiments – teacher and GPT-2 fine-tuning, augmentation data sampling, teacher logits generation, knowledge distillation and probing – are run in the environment of the University's Teaching cluster[11].

Each job uses its own one Nvidia GPU – either GeForce GTX TITAN X or GeForce RTX 2080 Ti – with 12-13GB of memory, and additional 30GB of RAM for use with CPU processes.

	CoLA	SST-2	Sara
teacher fine-tuning	~30min	~4h	~55min
GPT-2 fine-tuning	3min	31min	1min
augmentation data sampling	17h	15h	4h
teacher logits generation	~1h	~8h	~2h
LSTM_S training	~5h	~8h	~6h
BERT_S training	~15h	~26h	~22h

Table 4.5: The runtimes for all steps of knowledge distillation with augmented transfer datasets.

All important runtimes are reported in Tab. 4.5[12]. The reason why most steps take the longest on SST-2 is 1) the amount of training data (almost 10x more than for CoLA), and 2) the fact that sentences in SST-2 are longer than those in CoLA and Sara. Interestingly, even though LSTM_S and BERT_S are of similar size, the BERT model takes much longer to train – likely because it is much deeper.

Because of the role restrictions in the cluster, I cannot run more than 20 jobs at the same time. This has a significant impact especially on the time it takes to run the hyperparameter exploration experiments (see the next chapter). It is also the main reason why I do not – with a few exceptions – repeat experiments with varying random seeds for more robust results.

[11] `computing.help.inf.ed.ac.uk/teaching-cluster`.

[12] Note that the only processes that are parallelised are the augmentation data sampling and the teacher logits generation – both use 4 parallel threads, each with its own GPU with 6GB of memory. In logits generation, examples are processed in batches of 2048 in each thread.

4.5 Summary

In this chapter, I presented the high-level set up as well as the implementation details of all experiments. Importantly, the main outcomes of this exploratory work are intended to be insights, not improved performance scores. With most of the programming efforts going into adapting and integrating existing codebases, my original contributions are mostly intellectual: Using two architecturally different students side by side; using the probing suite of Conneau et al. (2018) for localisation of linguistic knowledge; using a new technique for extracting probing encodings; and later manually analysing the predictions made by the teacher and student models.

Chapter 5

Training student models

In this chapter, knowledge distillation is used to teach (train) student models BERT_S and LSTM_S from the fine-tuned teacher BERT_T. This is done separately on each of the three downstream tasks – CoLA, SST-2, and Sara. The objective is to obtain students which are small but perform well – relative to the teacher. Where possible, the student size is not increased above the initial dimensions outlined in Sec. 4.3.3 (LSTM_S) and Sec. 4.3.4 (BERT_S), which corresponds to keeping the number of trainable parameters at ~2.4 million.

As discussed in Sec. 4.1, my aim is not to find all the best possible student hyperparameters, but I still briefly explore some of them to gain an intuition for the reasonable ranges of values and for their behaviour in knowledge distillation. In particular, I find a well-performing training configuration of each student on CoLA, choosing based on the model's evaluation-set score. Then, on the remaining tasks, I use the same configuration, only tailoring a small number of parameters to the need of the concrete dataset at hand. (Most importantly, the student size is adjusted separately for each task because more difficult tasks may require larger (more complex) models for decent accuracy levels.)

After obtaining well-performing students for each task, these are briefly compared with the respective teacher in terms of model size, inference speed, as well as evaluation- and test-set scores.

5.1 Hyperparameter exploration

The initial exploration conducted on CoLA is restricted to these following essential hyperparameters in both students:

1. η – the learning rate. For LSTM_S, also the choice of a learning algorithm (the originally used Adadelta vs the more general Adam).
2. Learning rate scheduling: the warmup duration (in epochs) E_w of gradual warmup of η (Goyal et al., 2017), and the optional use of linear decay of η following after the warmup period (for an example, refer back to Fig. 4.2).
3. B – the minibatch size.

4. Embedding type and mode – word-level (initialised from word2vec) vs wordpiece (initialised from the respective BERT$_{\text{T}}$[1]), non-static vs multichannel[2].

The parameters are optimised one at a time, in the order they are enumerated above. At each step, the best value of one parameter is found and kept in all further steps. The explored values as well as the initial values (see Sec. 4.3) and the discovered best values are shown in Tab. 5.1. Note that the embedding type and mode is explored also later, separately for each task. For more details on how the individual parameters were chosen, see Sec. B.1 in Appendix B.

Most importantly, the LSTM student is found to outperform BERT$_{\text{S}}$ and converge much faster. Additionally, the LSTM prefers small batches (in line with the findings of Tang et al. (2019b)) and does not benefit from learning rate warmup (unlike BERT$_{\text{S}}$). Otherwise, the optimal training configuration seems to be similar in both students.

	explored values	
parameter	LSTM$_{\text{S}}$	BERT$_{\text{S}}$
η	**Adadelta ($\boldsymbol{\eta = 1.0}$)**, $\underline{\text{Adam}}$ with $\eta \in [5 \times 10^{-3}, 1.5 \times 10^{-3}, \underline{5 \times 10^{-4}}, 1.5 \times 10^{-4}, 5 \times 10^{-5}, 1.5 \times 10^{-5}, 5 \times 10^{-6}]$	$\underline{\textbf{Adam}}$ with $\eta \in [5 \times 10^{-3}, 1.5 \times 10^{-3}, \underline{\mathbf{5 \times 10^{-4}}}, 1.5 \times 10^{-4}, 5 \times 10^{-5}, 1.5 \times 10^{-5}, 5 \times 10^{-6}]$
η scheduling	$E_w \in [\underline{\mathbf{0}}, 5, 10, 15]$ + $\underline{\text{decay}}$/**no decay**	$E_w \in [0, 5, \mathbf{10}, \underline{15}, 20]$ + $\underline{\textbf{decay}}$/no decay
B	$[\underline{32}, \mathbf{50}, 128, 256, 512]$	$[32, 50, \underline{128}, \mathbf{256}, 512]$
embeddings	**word-level**/wordpiece + non-static/**multichannel**	word-level/**wordpiece** + **non-static**/multichannel

Table 5.1: The hyperparameter values explored on CoLA, one at a time, from top to bottom. In bold are shown the initial values. Underlined are the best values (for embedding mode and type, the best configuration is chosen separately for each task and is summarised elsewhere).

Following the initial hyperparameter exploration, the best embedding mode and type configuration is identified for each task. Observing the performance gap between the two students, I hypothesise that this may be due to the word-level embeddings in LSTM$_{\text{S}}$ being more suitable than the wordpiece embeddings in BERT$_{\text{S}}$.

The results of trying each embedding type combined with each embedding mode show that the multichannel mode is generally preferred[3], and that the best embedding type

[1] Initially, I experimented with LSTM$_{\text{S}}$ using word2vec embeddings (as in Tang et al. (2019a,b)) while the embedding layer of BERT$_{\text{S}}$ was randomly initialised. However, this poses a disadvantage for BERT$_{\text{S}}$ – it starts with no initial knowledge, unlike the BiLSTM student. To eliminate this disparity, BERT$_{\text{S}}$'s wordpiece embeddings were initialised with the teacher's wordpiece embedding parameters, and a trainable linear layer was added in the student to project these (high-dimensional) teacher embeddings to the smaller hidden dimensionality of the student $d_h = 204$. (Note that the token type embeddings and positional embeddings are not initialised from the teacher and hence do not require the linear transform for dimensionality reduction. Instead, these embeddings are added to the wordpiece embeddings inside BERT$_{\text{S}}$ after the wordpiece embeddings are dimensionality-reduced.) Even though the idea of initialising one model with another one's parameters is not new, to the best of my knowledge, I am the first one to initialise a BERT student in knowledge distillation in this way.

[2] The static mode (frozen embeddings, not allowed to be trained) was not tried, following preliminary experiments where freezing the parameters led to very poor performance.

[3] I.e. it is helpful to use an additional embedding matrix which is frozen during student training.

depends on the task (more details in Sec. B.2.1 in Appendix B). In particular, word2vec embeddings work slightly better for CoLA and SST-2, but are not preferred for Sara. This is likely due to Sara examples containing many mistyped words like "yesyesyes", which are treated as the general `UNKNOWN` word in word2vec, but are successfully broken down into smaller, meaningful units when using wordpieces. Thus, it may be preferable to use word2vec (or similar word-level embeddings) where the language is expected to be mostly clean, free of unusual or mistyped words (formal and semi-formal domains), while wordpieces provide a fallback alternative for informal domains.

While both of the 2.4-million parameter students perform very well on SST-2 and Sara (on par with the teacher), there continues to be a gap on CoLA: the teacher being far ahead, and LSTM_S outperforming BERT_S. To reduce this gap and to explore the effect of different student dimensions in general, I systematically vary the width and depth of each student – that is, the dimensionality of the hidden layers and internal representations, and the number of layers (encoder layers in BERT_S, BiLSTM layers in LSTM_S). On CoLA, the BERT_S is made up to 4x wider and 3x deeper, and LSTM_S is made up to 5x wider and deeper. On SST-2 and Sara, to explore how small the students can be to still achieve over 90% of their teachers' score, BERT_S is made up to 16x slimmer and 4x shallower, and LSTM_S is made up to 32x slimmer (originally with just one BiLSTM layer, it cannot be more shallow). (For details of the explored dimensions, see Tab. B.1 and Tab. B.2 in Appendix B; in particular, note that I had to manually reduce the learning rate for large BERT_S models to prevent gradient explosion.)

The results of the student size exploration (details in Sec. B.2.2) show that model width is, on these three tasks, more important than model depth. In particular, the performance gap between LSTM_S and BERT_S on CoLA is closed by increasing the width of the latter student (to roughly match the LSTM_S's layer width). On SST-2 and Sara, making the models slimmer affects their performance more than making them shallower. However, the results on CoLA suggest that LSTM_S may be too shallow for this difficult task and that using 2 and more hidden BiLSTM layers is beneficial (compare with BERT_S, which uses 5 hidden layers by default, and further increasing this does not help).

5.2 Discussion and summary

Tab. 5.2 summarises the hyperparameters chosen for each student on each task, and compares all models in terms of their size, prediction speed[4] and score. Clearly, CoLA is more difficult of a task than SST-2 and Sara: In the student size exploration, even models with over 100 million parameters achieve only below 75% of the teacher's score. As a compromise between accuracy and model size, I choose for analysis students that are 4.5x (BERT_S) and 22x (LSTM_S) smaller than the teacher and both achieve similar scores. On SST-2 and Sara, on the other hand, the ~2.4-million-parameter students, being ~140x smaller than BERT_T, reach comparable accuracy. The students can be even smaller while staying above 95% of the teacher's score: E.g. LSTM_S on SST-2 can be 14,000x smaller (64x slimmer than the 2.4-million-parameter version), and BERT_S can be 2000x smaller (3x slimmer and 2x shallower than the 2.4-million-parameter version). The students are

[4]Timed on a laptop with an Intel Core i7-6600U CPU, i.e. not using a GPU.

	model	dimensions	training	embed.	size	RPE	eval	test
CoLA	BERT_T	$L=24,\ d_h=1024,$ $d_I=4096,\ A=16$	$B=36,\ \eta=5\times10^{-5},$ $E_w=0.3$, decay	piece	340M	750ms	59.9	54.6
	BERT_S	$L=5,\ d_h=816,$ $d_I=3000,\ A=12$	$B=128,\ \eta=7\times10^{-5},$ $E_w=15$, decay	word, multi	76.4M	100ms	45.0	29.8
	LSTM_S	$L=2,\ d_{LSTM}=600,$ $d_{FC}=800$	$B=32,\ \eta=5\times10^{-4},$ $E_w=0$, decay	word, multi	15.4M	2.3ms	44.2	27.9
SST-2	BERT_T	$L=24,\ d_h=1024,$ $d_I=4096,\ A=16$	$B=36,\ \eta=5\times10^{-5},$ $E_w=0.3$, decay	piece	340M	750ms	91.5	93.1
	BERT_S	$L=5,\ d_h=204,$ $d_I=750,\ A=3$	$B=128,\ \eta=5\times10^{-4},$ $E_w=15$, decay	word, multi	2.42M	12ms	89.3	87.8
	LSTM_S	$L=1,\ d_{LSTM}=300,$ $d_{FC}=400$	$B=32,\ \eta=5\times10^{-4},$ $E_w=0$, decay	word, multi	2.41M	1.0ms	91.2	92.2
Sara	BERT_T	$L=24,\ d_h=1024,$ $d_I=4096,\ A=16$	$B=36,\ \eta=5\times10^{-5},$ $E_w=1$, decay	piece	340M	750ms	87.5	88.3
	BERT_S	$L=5,\ d_h=204,$ $d_I=750,\ A=3$	$B=128,\ \eta=5\times10^{-4},$ $E_w=15$, decay	piece	2.43M	11ms	87.1	86.4
	LSTM_S	$L=1,\ d_{LSTM}=300,$ $d_{FC}=400$	$B=32,\ \eta=5\times10^{-4},$ $E_w=0$, decay	piece, multi	5.90M	0.68ms	86.5	86.4

Table 5.2: Essential information about the students and teachers on each downstream task. The model size is in millions of trainable non-embedding parameters. The embedding type ("embed.") is "word" (word2vec) or "piece" (wordpiece), the mode is either "multi" (multichannel), or the default non-static mode where not explicitly stated. The inference runtime per example (RPE) is calculated by measuring the time for processing the entire evaluation set in batches of 256 (previously transformed from text into numerical form), then turned into time per single example and reported. Evaluation-set ("eval") and test-set ("test") scores are reported using the appropriate metric (accuracy, MCC, F1_{micro}). The student size differs between SST-2 and Sara due to the higher dimensionality of wordpiece embeddings (1024) compared to word2vec (300).

also much faster than the teacher models, with LSTM_S being particularly fast[5] – up to 1100x faster than BERT_T.

There is evidence of model width being the key dimension, with small model depths being sufficient for decent performance levels. Possibly, models like the 12- and 24-layer BERTs released by Devlin et al. (2019) are unnecessarily deep for tasks like intent classification or even grammatical acceptability. Thus, making the models shallower is one way of compressing and accelerating them (in line with Sanh et al. (2019) who created well-performing BERT student shallower but not slimmer than BERT_Base).

There are several differences between BERT_S and LSTM_S. Notably, the LSTM student converges much faster, but works best with smaller minibatches, which makes training slower compared to large batches. The BERT student is more sensitive to the learning rate values and these need to be significantly reduced for larger BERT_S sizes. Otherwise, the models are not too sensitive to hyperparameter choices, and the configuration chosen on CoLA works well when used in students trained on SST-2 and Sara.

[5] LSTM_S may be faster than BERT_S partly due to different input feeding strategies; while for BERT_S all input examples are padded to the model's maximum sequence length of 128, for LSTM_S, I only pad all examples within a batch to the length of the longest example, which leads to more compact batches.

In knowledge distillation, the teacher's knowledge enters the student in the top layer (where feedback is received during training, with error gradients "trickling down" into the lower student layers). The provision of trained embeddings to a student creates the opposite (and complementary) flow of knowledge: from the bottom up, as the knowledge captured in the embeddings propagates into the rest of the model. I showed that while both word-level and wordpiece embeddings work well with both student architectures, certain downstream tasks (here CoLA and SST-2) benefit from the higher-quality word-level representations while others like Sara need the flexibility of wordpiece embeddings. It would be interesting to see how well word-level embeddings fine-tuned as part of the teacher model would perform.

Besides leaving the embedding layer to be further trained during knowledge distillation, I observe the usefulness of keeping another – frozen – copy of the embeddings[6]. In other words, the student benefits from having access both to the original embeddings and to the embeddings trained as the student learns.

The 9 models described in Tab. 5.2 – one teacher and two students for each task – are further analysed in the remainder of this work.

[6]This corresponds to the multichannel embedding mode.

Chapter 6

Analysing the models

In this chapter, probing and prediction analysis are used to analyse, interpret and compare the teacher and the student models for each downstream task. The aim is to produce insights into the nature of the downstream tasks, the models, and knowledge distillation.

6.1 Probing

By probing, in this section, I find that model initialisation plays a key role in student learning; localise different linguistic skills in the models; and point out possible limitations of the probing suite.

Tenney et al. (2019a) recently showed that a typical text processing pipeline can be identified in BERT. In the probing suite used here (Conneau et al., 2018), the pipeline steps are represented from the simplest ones to the most complex ones; from extracting surface properties of input such as sentence length (probing task `Length`) to detecting broken semantics (tasks `OddManOut` and `CoordinationInversion`); re-visit Sec. 3.3 for more details on each probing task. Here, probing is used to trace language knowledge which enters models in various ways – as pre-trained knowledge (in the teacher BERT before fine-tuning), via trained embedding parameters (in both students), or the knowledge flow from $\mathrm{BERT_T}$ into students during knowledge distillation. While Conneau et al. apply probing only to the last layers of their models, I probe different layers[1] in order to also localise the language knowledge within models (similar to Tenney et al.).

Probing results are shown in Fig. 6.1 (teacher models) and in Fig. 6.2 (students). In general, students achieve lower probing scores than their teachers, especially on the difficult, semantical tasks (`OddManOut`, `CoordinationInversion`, partly `BigramShift`), which require good sentence-level understanding, not just word-level knowledge found in embedding parameters.

Several architectural differences between the models are reflected in probing results. In the deep teacher models, only the last layers change in fine-tuning (compare fine-tuned

[1] As discussed in Sec. 4.3.6, in the case of $\mathrm{LSTM_S}$, I am only able to probe the final representations due to the way LSTMs are implemented in PyTorch.

teachers with the pre-trained $\text{BERT}_{\text{Large}}$), while the earlier layers act like downstream-task-agnostic general feature extractors[2]. In the shallow LSTM_{S}, the BiLSTM layer also likely serves as a general feature extractor – its probing score is comparable across the downstream tasks, and is often better than that of the downstream-task-specific last layer of BERT_{S}. The performance gap between the students on `WordContent` may be linked to the lack of residual connections in LSTM_{S} – in the BERT student, these connections enable easy copying of input into higher layers. Lastly, the recurrent LSTM student architecture may be more suited for order- and length-sensitive tasks (`Length`, `BigramShift`, `CoordinationInversion`), as the results on SST-2 and Sara show.

Where the downstream task does not require sophisticated linguistic skills (SST-2, partly also Sara), the language knowledge is mostly lost or not acquired in both teachers and students[3]. On CoLA, on the other hand, the linguistic skills are mostly retained/acquired or even slightly improved in the later layers of both teachers and students

Additionally, Fig. 6.1 confirms that surface skills (`Length`, `WordContent`) are found in the early teacher layers, syntactic skills (`Depth`, `TopConstituents`) are in the middle layers, and semantic skills (`OddManOut`, `CoordinationInversion`) concentrate in the final layers.

[2] Perhaps these layers could be frozen in order to make fine-tuning faster.

[3] Only on `WordContent`, the CoLA teacher is outperformed by the Sara teacher; I attribute this to many Sara intents being recognisable by characteristic keywords (e.g., examples of the intent `affirm` typically contain "yes" or "okay"), which motivates the Sara teacher to learn to "remember" exact input words.

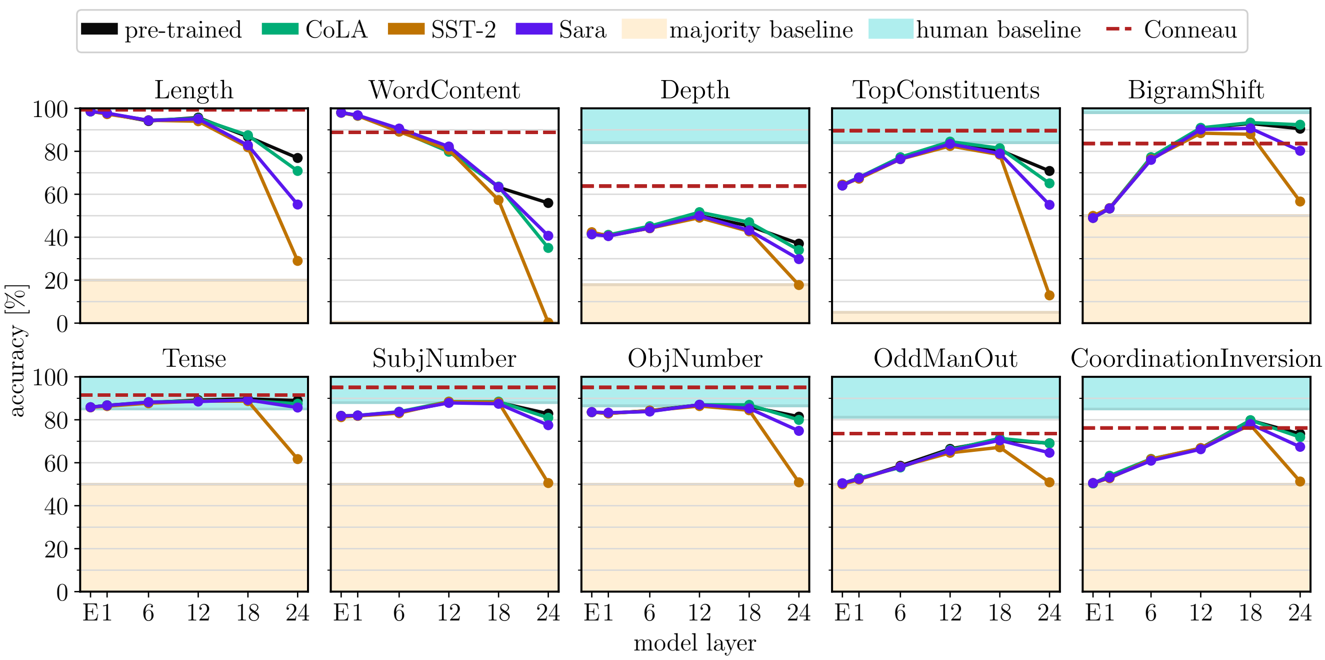


Figure 6.1: Probing results for the pre-trained BERT and the teacher BERT models. Probing was applied to encoder layers 1, 6, 12, 18, and 24, and to the embeddings (layer "E") extracted just before the first encoder layer. The two baselines, reasonably bounding the expected model performance from below and from above, are the majority-class baseline and the human performance baseline; additionally, the best model scores are shown ("Conneau") – all baselines as reported by Conneau et al. (2018).

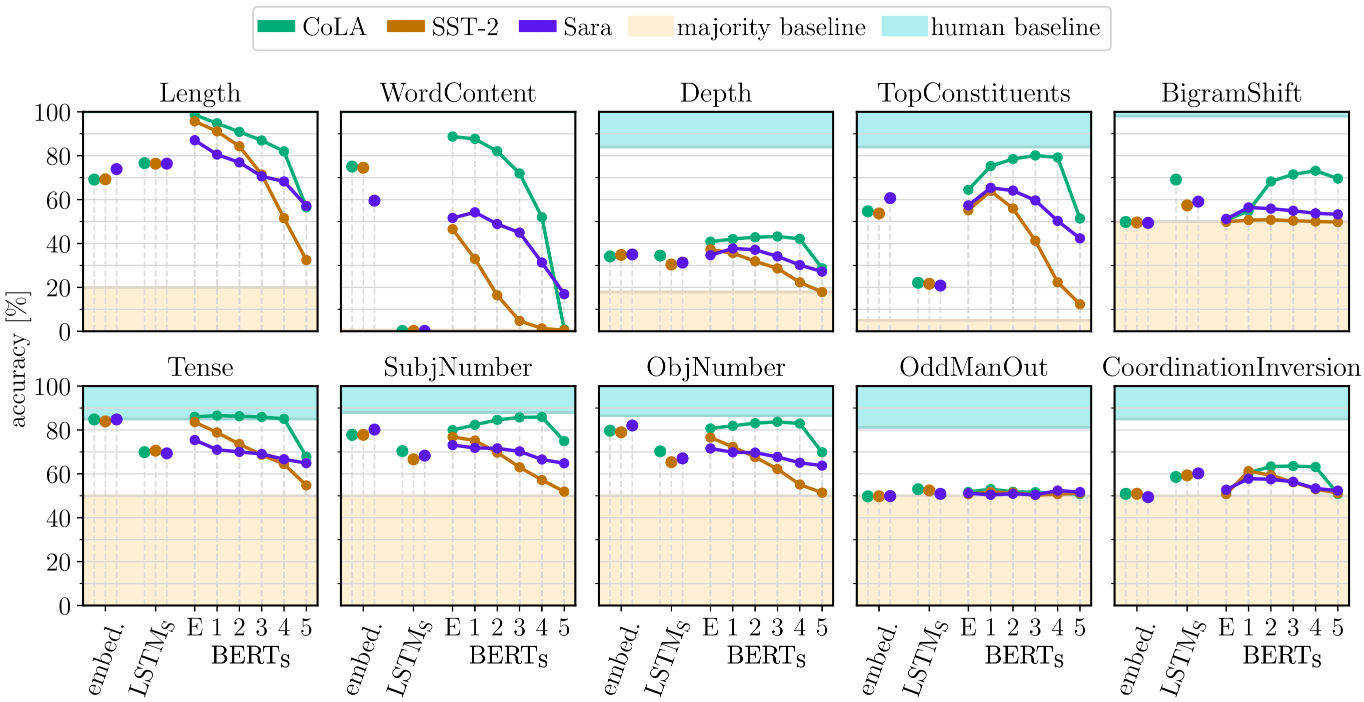


Figure 6.2: Probing results for the best student models. For comparison, results achieved just by using the embeddings (taken before student training) are shown as well ("embed.") (the average pooling strategy was used to construct probing sentence encodings from the individual embeddings of a sentence's words/wordpieces). For LSTM_S, the probing encodings were extracted only after the last LSTM layer; for BERT_S, they were extracted after each encoder layer (1-5) and before the first layer ("E"). The majority-class baseline and the human performance baseline are shown, same as in Fig. 6.1.

The results show important effects of initial "provision of knowledge" to students via trained embedding parameters. In BERT_S, often only the bottom layers achieve good scores (see `TopConstituents`, `Tense`, `SubjNumber` and `ObjNumber` in Fig. 6.2)[4], which reflects the knowledge "leaking" from the embeddings up through the model. This knowledge captured solely by the embeddings is significant (see the "embed." results in Fig. 6.2), and Fig. 6.3 shows that when it is not given to students before training[5], the embedding-reliant skills (`Tense`, `SubjNumber`, and `ObjNumber`) worsen, with the bottom layers of BERT_S no longer achieving good scores. In light of these results, and by showing that a very simple rule-based morphology-guessing model[6] can achieve decent scores ("morph. guess" in Fig. 6.3), I argue that these tasks should be used carefully as indicators of semantic skills.

All in all, probing can provide useful insights but it is important to interpret the results correctly: A high score might not mean that the model layer *learnt* the skill. Perhaps,

[4]This could be verified for the LSTM student if it had more layers and if it was possible to probe all of them separately.

[5]I.e. when all student parameters are initialised randomly from scratch.

[6]This model is based on the observation that, in English, just knowing if *any of the words* in a sentence are in the plural form (dominantly marked by the suffix "-s/-es") is a decent proxy of whether *the subject/object* is in the plural form, and similarly with verb tense (present/past) marked by the suffix "-d/-ed". Note that such morphological information can be captured in word2vec, as observed by Gieske (2017).

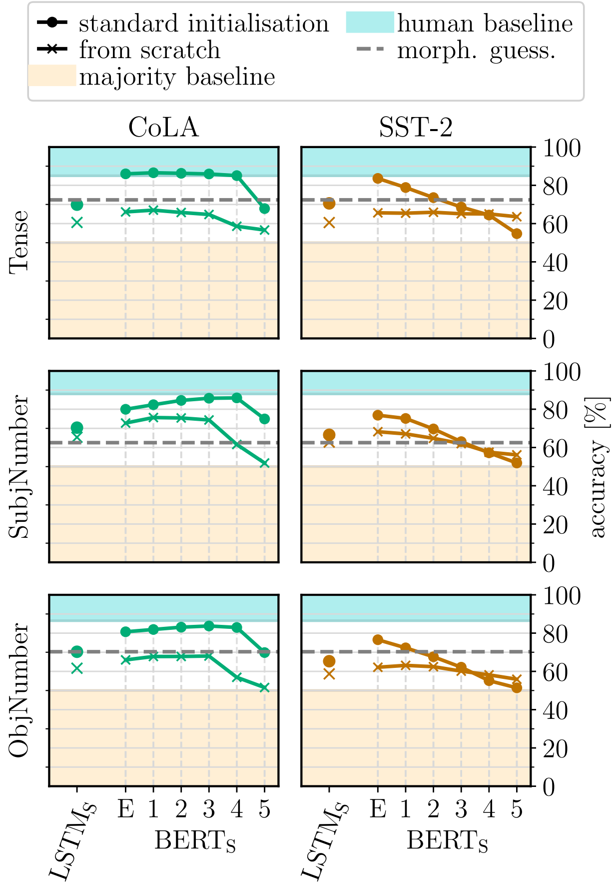


Figure 6.3: Probing results for selected downstream and probing tasks, comparing students initialised in the standard way (with embeddings from word2vec) with students initialised randomly and trained from scratch. Bounding the expected model performance from below and from above are again the majority-class baseline and the human performance baseline, same as in Fig. 6.1. Additionally, the performance of a simple "morphology-guessing" model is shown, which transforms any input sentence into "0" or "1" depending on whether any of the sentence's words end in "d" or not (for `Tense`) or in "s" (for `SubjNumber` and `ObjNumber`).

knowledge was present before training either in that layer or in a neighbouring model component. Last model layers are also misleading because they focus on task-specific knowledge, and only show linguistic skills if the downstream task explicitly needs them (such as CoLA).

6.2 Analysing the models' predictions

In this section, by inspecting the models' evaluation-set predictions, I observe that the most sophisticated skills are not transferred well into the students; that both students make similar mistakes; and that the LSTM student can better mimic the teacher.

In inspecting the predictions, both correctness and cofidence is considered[7]. Here, I define

[7]Certainly, a confident incorrect prediction is not the same as a very unconfident decision which also happens to be incorrect.

the confidence of a prediction as the probability assigned to the predicted class[8]. While the analysis comprises mostly qualitative, manual inspection of sentences accompanied by predicted labels and prediction confidences (see Fig. 6.4), where possible, quantitative, aggregated results are also presented.

confidence	sentence	true	predicted
CONFIDENT MISTAKES			
0.999	i am an opioid addict	out_of_scope	enter_data
0.998	chatfuel	switch	enter_data
0.998	i am qq	out_of_scope	enter_data
0.997	toodle-oo	bye	greet
0.996	how many candles were on your last birthday cake?	ask_howold	out_of_scope
0.995	how do you learn	out_of_scope	how_to_get_started
0.993	wit	switch	enter_data
0.992	i want to know how can buld my own bot	how_to_get_started	out_of_scope
0.991	how many languages does spacy support?	out_of_scope	ask_faq_languages
0.990	ok let's start	affirm	how_to_get_started
UNCONFIDENT MISTAKES			
0.118	you originated through what means?	ask_howbuilt	ask_whatisrasa
0.136	i would just like to have the link for the community	ask_faq_what_is_forum	signup_newsletter
0.153	tell me what's your skill	ask_whatspossible	out_of_scope

Figure 6.4: Example of the interface used for inspecting the predictions, in this case the predictions of LSTM_S on the Sara task.

To give the analysis a logical structure, I propose these three objectives, each realised as a number of analysis tasks (see Tab. 6.1): 1) characterising the teacher and student models individually; 2) characterising the differences between the models; and 3) characterising knowledge distillation through its limitations.

When choosing examples for inspection, I select the most extreme cases, i.e. taking the most/least confident hits in P1 or the examples with largest confidence gap between the relevant models (P5, P8). Where such sorting is not possible, examples are selected randomly from all suitable ones (P4, P7). In all cases, 10 examples are selected for inspection in order to reasonably constrain the task[9]. In P4, examples are selected such that they are classified correctly by BERT_T[10].

Tab. 6.2 summarises the observations from each analysis task. In general, examples referred to as "easy" are mostly simple sentences on CoLA (*the witch poisoned the children.*), sentences with clear sentiment on SST-2 (*a gorgeous, witty, seductive movie.*), and

[8]For binary classification, the minimum confidence is 0.5 and the maximum is 1.0. A possible limitation of this definition is that confidence may be generally lower on tasks with many classes, because softmax-produced probabilities are always above-zero for every single class.

[9]This still means inspecting 360 sentences for P1: For each downstream task and each model, the 10 most confident hits and mistakes, as well as the 10 most unconfident hits and mistakes.

[10]This is used as an indicator of the particular example being not too difficult to classify correctly, meaning that both students have a reasonable chance of being correct, even though one of them still makes an incorrect prediction.

	point of focus	rationale
individual models	P1: individual examples predicted correctly (hits) and incorrectly (mistakes); separately the most confident and unconfident cases	understanding individual students' strengths/weaknesses
	P2: average confidence: overall, separately on mistakes and hits	
model differences	P3: comparing average confidence levels of models	understanding differences between students in terms of their skills and confidence patterns
	P4: individual examples predicted correctly by only one student (incorrectly by the other one)	
	P5: individual examples predicted confidently by only one student (unconfidently by the other one)	
	P6: average overlap of the mistakes and hits of different models	
knowledge distillation	P7: individual examples predicted correctly by the teacher and incorrectly by both students	understanding the skills not learned via distillation
	P8: individual examples predicted confidently by the teacher and unconfidently by both students	

Table 6.1: The structure of the prediction analysis: The points of focus for each of the three main areas in which insights are mined from the model predictions.

sentences with clear keywords on Sara (*the bot speaks **spanish***[11]). Difficult examples may require detailed understanding, e.g. recognising the broken semantics in *my heart is pounding me.* (CoLA).

[11] Names of several languages appear extremely often in the enter_data intent examples.

P1	Confident hits: Easy examples from dominant classes (acceptable for CoLA, enter_data for Sara; see Fig. A.1 for detailed class distribution in each dataset). Unconfident hits/mistakes: Difficult examples; with mixed positive and negative words (SST-2); without characteristic keywords (Sara). Confident mistakes: Examples with questionable labels, strong words opposing the overall sentiment (SST-2); misleading keywords characteristic of an incorrect intent (Sara); examples where recognising unacceptability requires semantic understanding (CoLA).
P2	On average, models are more confident on hits than on mistakes (see Fig. 6.6).
P3	On average, BERT_T is slightly more confident than either student (may be due to scoring more of the [confident] hits, especially on CoLA) (see Fig. 6.6).
P4	All examples are long and difficult, BERT_T is unconfident as well as one or both students (not necessarily the incorrect one). On Sara, several examples where BERT_T better understands meaning and is not mislead by keywords, but one student is.
P5	All the examples are difficult, with all models often unconfident or even mistaken. On SST-2, the LSTM_S is unconfident often when the teacher is unconfident (the same correlation is not observed for BERT_S). Fig. 6.5 quantitatively confirms this and shows moderate correlation between all model pairs' confidences on Sara (and mostly weak correlation on other downstream tasks).
P6	Students are relatively unique in their mistakes: On CoLA and SST-2, ~60-70% of their mistakes are shared by both students. On Sara, this sharing is above 80%. The architecturally different LSTM_S learns to copy BERT_T's behaviour more closely than the BERT student (mostly in terms of copying the teacher's mistakes). [For detailed quantitative results, see Fig. C.2 in Appendix C.]
P7	Very difficult examples; classified unconfidently (and mostly incorrectly) by both students, and mostly correctly by BERT_T. Overall, they demonstrate the teacher's superiority on challenging sentences, e.g. recognising *bill's story about sue and max's about kathy both amazed me.* as acceptable (CoLA) or *you live around here?* as ask_wherefrom (Sara).
P8	Similar to P7; complicated examples that require understanding of semantics (CoLA, Sara) and of mild sentiment possibly expressed in metaphors (SST-2), e.g. *my heart is pounding me* (CoLA, unacceptable), *fuck yeah!* (Sara, affirm; misclassified by both students as handleinsult).

Table 6.2: Main observations from prediction analysis, individually for each analysis task from Tab. 6.1.

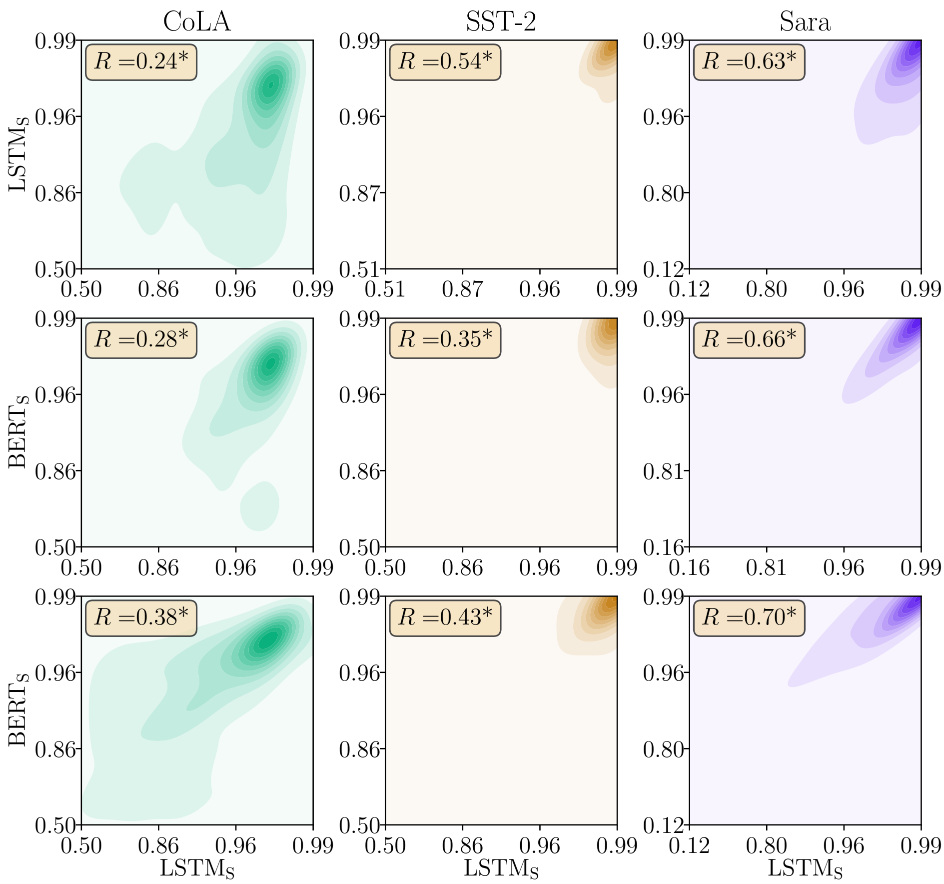


Figure 6.5: Correlation between the prediction confidences of different models, measured as the Pearson correlation coefficient R. The "*" marks statistically significant correlation (for $p < 0.05$).

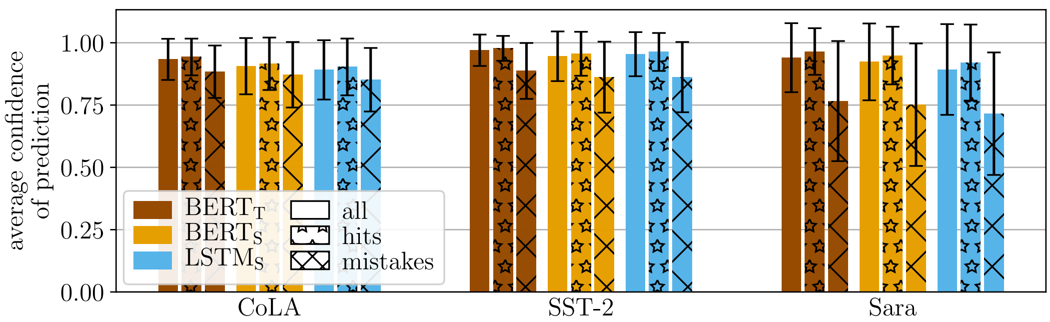


Figure 6.6: The average confidence of evaluation-set predictions. Standard deviation errorbars are shown.

To further aid my understanding of predictions on CoLA, lastly, I employ an experimental approach: I let all models classify numerous perturbed variants of 21 interesting CoLA sentences which were originally predicted correctly by BERT$_\mathrm{T}$ but incorrectly by one or both students[12]. This produces further evidence of students not being sensitive to valid/broken sentence semantics, see Tab. 6.3. In addition to "allowing" John to be

[12] The complete list of the sentences and their perturbed versions is in Tab. C.1 in Appendix C.

a tree, the students are also found to be sensitive to concrete word choices where these should not matter, e.g. *most of the fruit is ripened.* is classified differently from *most of the fruit is spoiled.*

	label	BERT_T	BERT_S	LSTM_S
my heart is pounding me.	✗	✗ 0.89	✓ 0.96	✓ 0.88
my heart is pounding.	✓	✓ 0.98	✓ 0.97	✓ 0.98
my heart is beating me.	✗	✗ 0.92	✓ 0.93	✓ 0.98
my heart is beating.	✓	✓ 0.98	✓ 0.93	✓ 0.98
we believed john to be a fountain in the park.	✗	✗ 0.92	✓ 0.97	✓ 0.98
we believed john to be a bench in the park.	✗	✗ 0.93	✓ 0.96	✓ 0.98
we believed john to be a musician in the park.	✓	✓ 0.98	✓ 0.98	✓ 0.98

Table 6.3: Two example CoLA sentences, each followed by my own perturbed variants. ✓ = acceptable, ✗ = unacceptable. Confidences are shown next to model predictions.

6.3 Summary

Probing and prediction analyses were applied to the teacher and student models, each shedding light on a different aspect.

While probing results show student models being less linguistically competent than their teachers, this does not automatically imply that the students failed to properly learn what they should. Instead, I point out that the teachers (and, more generally, models initialised from trained parameters) can contain linguistic knowledge not needed for the downstream task at hand. Additionally, observations from probing quantitatively confirm that CoLA as a task requires many different linguistic capabilities, whereas SST-2 and Sara do not. As for differences between the different student architectures, LSTM_S may be more sensitive to the notions of length and order in input sentences. As a limitation of the probing suite of Conneau et al. (2018), I point out that three of the tasks are handled relatively well even by simple bag-of-embedding models, as well as by trivial rule-based morphological models.

The mostly qualitative prediction analysis helped to characterise sentences that are easy or difficult for the models. The LSTM student was shown to better copy the teacher's behaviour. Still, both students fail to acquire the most sophisticated skills of their teachers – notably, the students exploit easy cues such as keywords, but have limited sense of sentence semantics.

Chapter 7

Overall discussion, conclusions and future work

In this chapter, findings from all of this work are broadly discussed, before overall conclusions and ideas for future research are presented. In the discussion, I focus on knowledge distillation in an applied context where model size and speed is to be optimised. The model analysis approaches are viewed as tools for understanding and subsequently improving knowledge distillation in practice.

7.1 Distilling BERT into tiny models

With a lot of recent research on knowledge distillation for compressing BERT, one should carefully recognise that different works have different objectives. One stream of work aims to retain all of BERT's accuracy (Sanh et al., 2019; Sun et al., 2019b; Jiao et al., 2019), but the size compression ratio is 2-10x, which may still produce models too large and slow for many applications. Another stream (Tang et al., 2019b,a) explores the limits of model compression using knowledge distillation at the cost of losing some of BERT's accuracy. My work is closer to the second stream, originating from previous attempts at heavily accelerating BERT (Sucik, 2019). Adopting an applied mindset, I also focus on how the distillation process can be tailored for individual tasks, instead of proposing a universally well-performing student. While I observe that one student configuration can work well across several tasks, several decisions are best made on a case-by-case basis, as discussed in what follows.

The student model dimensions should reflect the task at hand. In particular, in research and applications aiming to compress models as much as possible, it may be preferred to tailor the student size individually to each task, as opposed to benchmarking a fixed-size architecture on a diverse suite like GLUE. I showed that on easy tasks like SST-2 or Sara, a student can be several thousand times smaller than the $\text{BERT}_{\text{Large}}$ teacher and still retain over 95% of the teacher's score. However, CoLA, as a complex linguistic task, requires students to be both wider and deeper, and still they only achieve ~75% of the teacher's score. Moreover, I observe that it is the student width, not depth, that plays a key role: Increasing the number of BiLSTM or Transformer layers beyond 1-3 generally does not improve scores, but accuracy degrades when the layer width is below 200-300 (with CoLA requiring ~2-3x wider representations).

Besides the student size, model initialisation from trained parameters is important – as a way of giving students language knowledge to start with. Such initialisation makes sense especially in the lower layers, which 1) tend to learn task-agnostic features (thus, general trained parameters can be re-used across tasks), and 2) are located far from the top layer through which the teacher's knowledge "enters" (which makes learning in these lower layers more difficult). I observe that the choice of embeddings (word-level vs wordpiece) can improve performance and is also possible to reason about, given some knowledge of the data[1]. While initialisation from trained parameters is most easily done in the embedding layer, ideally, this would be extended to multiple early layers[2]. In the future, probing could be used to explore the flow of teacher's knowledge in distillation variants that force students to mimic internal teacher layers (e.g. Jiao et al. (2019); Sun et al. (2019b)). When possible, copying the teacher's encoder layer parameters directly into the student is also an attractive option (Sanh et al., 2019); however, this requires the student to be as wide as the teacher, which can be undesirable.

Last but not least, the choice of student architecture is important. While most studies on distilling BERT work towards smaller BERT versions, there is no reason to believe that other student architectures should be inferior. While in my experiments both students tend to perform comparably well, I observe the BiLSTM student to be more sensitive to order and length phenomena. Recently, Huang et al. (2020) showed that the two architectures can complement each other – by combining them into one, they improved on BERT's accuracy. In practical applications, inference speed can make one architecture preferred, perhaps because its concrete implementation is more optimised. For instance, LSTMs can process variable-size batches whereas BERT requires all inputs to be padded to a fixed maximum sequence length; this way, LSTMs can process shorter inputs faster, but BERT can not.

7.2 What can models tell us

Coming up with an idea for improvement; building a new model; evaluating it on a benchmark like GLUE – these are the typical steps shared by many works on improving language models. I am of the opinion that trained models can reveal a lot about themselves and make suggestions for further iterative improvements. In this work, I demonstrate the use of two complementary analysis approaches with this aim.

Probing tasks can be used beyond the purposes they were originally meant for. Conneau et al. (2018) propose tasks for quantifying the language knowledge present in the last layer of a sentence encoder model. Tenney et al. (2019b,a) inspect individual model layers and localise different linguistic capabilities within BERT. I combine the above approaches and, additionally, use probing to "track" language knowledge as it enters and propagates

[1] Word-level embeddings being more suitable for domains with decent language, wordpieces more useful where many unknown words (slang, mistyped and similar) are expected.

[2] This is not commonly done, as these parameters are specific to each architecture, unlike the token-level embeddings, which are highly re-usable.

through the models before and during training. This way, I characterise the impact of initialising the students' embedding layer from trained parameters, and the loss of general language knowledge in the top layers of the pre-trained BERT when this is fine-tuned on concrete downstream tasks.

Both probing and inspecting the models' predictions helps to describe the teacher's skills not learnt well by students. In particular, I observe that semantic skills are diminished in the students even when the downstream task clearly requires them (CoLA). As a remedy, distilling this complex knowledge from the teacher's intermediate layers (Jiao et al., 2019; Sun et al., 2019b) could be tried, with the effects monitored by student probing.

Probing can motivate task-specific model adjustments. On CoLA, which is found to leverage complex linguistic skills from later model layer, the students can be made deeper to facilitate the learning of complex, semantic representations. On Sara, inspecting the model's predictions reveals tendency for learning characteristic keywords. Depending on whether this behaviour is desirable or not, residual connections in students can be added/removed to make internal input-copying easier or harder.

Considering the confidence of predictions provides richer insights into model behaviour. I observe that unconfident predictions correspond to challenging examples, whereas confident predictions are made on easy examples (classified correctly) and on "tricky" misclassified examples where models are misled often by a single word. Calculating the correlation between two models' prediction confidences can quantify the extent to which models behave similarly in their perception of examples as easy or difficult. This way, I observe that the teacher and student models diverge the most on CoLA and SST-2, but behave similarly on Sara examples.

Both probing and prediction analysis have their limitations. Correct interpretation of probing results is difficult: A high probing score does not imply that the model actively learnt the skill in question. Instead, the knowledge may have been present in trained embeddings, or acquired from the transfer dataset which reflects the teacher's knowledge but not necessarily the task's needs. Furthermore, just because a probing task is aimed for measuring a specific skill, it may not measure this well – for instance, a semantic task may still heavily rely on easy lexical cues, or may represent only a very narrow part of general semantic knowledge. Inspecting a model's predictions has its own caveats – in particular, it is difficult to grasp why exactly a model misclassifies a given sentence. While I show that this can be sometimes clarified by testing the model on numerous hand-crafted perturbations of the original sentence, this approach is extremely laborious.

7.3 Conclusions

In this work, I have explored the use of teacher-student knowledge distillation for compressing the large BERT language model into architecturally different, smaller models, separately on different sentence-classification tasks. With the aim of extreme model compression, I adopt the approach of first fine-tuning BERT on a specific task and subsequently distilling it into student models.

My findings show that easier tasks like SST-2 (binary sentiment classification from movie reviews) and Sara (57-way intent classification of human messages from human-bot conversations) can be successfully handled by student models several thousand times smaller and faster than the $\text{BERT}_{\text{Large}}$ teacher model. However, these tasks are very easy to start with, and applying BERT to them may be questioned. On the very challenging task of linguistic acceptability judgement (CoLA), standard knowledge distillation cannot bridge the gap between the teacher and the students, and a more sophisticated way of knowledge transfer may be needed, especially to help the students acquire semantic skills.

Working with different student architectures and on different downstream tasks is relatively easy. In particular, roughly the same hyperparameter configuration can be re-used. However, the students' depth is best adjusted on each task (the more complex the task, the deeper the students), as is the choice of pre-trained embeddings (word-level vs wordpiece). Both a bidirectional LSTM student and a down-scaled BERT student are found to work similarly well, with the former one learning to copy the teacher's behaviour more closely, despite the architectural disparity.

I show that probing the models for specific linguistic knowledge as well as inspecting the models' predictions can be used to mine various insights about the models, the tasks, and about knowledge distillation. While the first approach quantitatively measures different linguistic skills possessed by a model, the latter approach can provide useful concrete examples for further analysis. In general, both students are found to lack especially the complex semantic understanding possessed by the teacher, and challenging sentences are presented on which the teacher succeeds but the students fail. Further, probing is shown to usefully trace the different sources of knowledge in models, producing insights which can be used to adjust the student architecture and initialisation.

Overall, I have produced useful insights both by using knowledge distillation in different settings and by analysing the trained teacher and student models.

7.4 Directions for future work

While the approach of applying knowledge distillation to large Transformers is new and many things are yet to be properly researched, my focus is primarily on analysing the technique and the models it produces, the main aim being better understanding.

As the biggest downside of prediction analysis, I identify my uncertainty in identifying why a model misclassified a given sentence. While I show that this can be partly addressed by perturbing the sentence and effectively testing different hypotheses, the approach is very laborious. Naturally, being able to easily and quickly interpret a model's predictions is desirable. In future work, inspecting predictions could hugely benefit from automatically generated "saliency maps", i.e. highlighting the input tokens which are most responsible for the particular prediction outcome. In practice, such saliency highlighting can be achieved either by using attentional weights like in the original self-attention (Lin et al., 2017), or, where the student doesn't use attention, by more general methods (several of them recently built into an NLP model interpretation toolkit by Wallace et al. (2019)). Additional automated approaches include masking out one or more words at a time and scoring such incomplete sentences, or, with sequential models such as LSTMs, scoring progressively larger parts of the input to observe how the model's beliefs evolve while "reading" the sentence.

Observing that probing can be used for effectively "tracking" the flow of language knowledge, one promising research direction is making such tracking more systematic. Importantly, creating a proper methodology around the use of probing tasks is desirable. Such methodology should facilitate more careful interpretation of probing scores, in particular by controlling for language knowledge which is present in the model for reasons other than the hypothesised one (e.g. as residual knowledge from pre-trained parameters). It is desirable to know how and why certain knowledge was acquired by a model, not just where and how much of this knowledge is possessed by the model. As another improvement of the probing approach, it could be applied not just to layer outputs, but also to the self-attention heads' outputs in Transformer models. This could either use existing general probing suites like that of Conneau et al. (2018), or adopt a more bespoke approach similar to Clark et al. (2019) who probe the heads for relational language knowledge.

While the probing tasks used today rely on classification tasks[3], a different approach to probing may be to quantify the general sensitivity of a model to a particular *concept* (such as the concept grammatical tense). For this purpose, the recently proposed approach of Kim et al. (2018) could be used, which enables measuring any neural model's sensitivity to any concept representable by a collection of input examples. In the NLP domain, several concepts are especially easy to gather examples for: the concept of sentence length (exemplified perhaps by a collection of long sentences), the concept of sentence type (e.g. a collection of interrogative sentences, identified automatically by the final "?"), the concept of tense (curated with the help of an automatic lexical parser), the concept of formal or informal language (as warranted by a particular text source), etc. Ideally, this would enable probing tasks to be more easily extended beyond linguistic skills. Additionally, it is interesting to know a model's general sensitivity to, say, the concept of tense, and compare this with the model's score on a binary tense-classification task.

Naturally, further exploration could focus more directly on improving knowledge distillation as such. In particular, since student training takes long on the large transfer datasets, it would be interesting to try varying the amount of augmentation data while observing not just the students' scores on downstream tasks, but also the various language knowledge acquired. Additionally, I observe that the GPT-2-generated augmentation sentences (especially on SST-2 and Sara) are often non-sensical (e.g. *what management nyy* or *done str prevents them*). Generating more credible input examples – by fine-tuning GPT-2 for longer, or by scoring the generated sentences for grammaticality by another model – could improve knowledge distillation. The importance of the presence of the original training data in the transfer dataset is also unclear; this could be reduced or, on the other hand, amplified (by including the data multiple times), while observing the effects of such manipulation.

Finally, other model compression techniques can be explored, similarly to knowledge distillation. One attractive approach is model pruning which removes individual weight connections or neurons (Han et al., 2016; Sajjad et al., 2020), or even entire attentional heads (Michel et al., 2019) or model layers (Mao et al., 2020), often without significantly

[3] For instance, a model is tested on its ability to distinguish between the past and present tense of a sentence's main verb.

affecting the model's accuracy. The question is if and what linguistic skills are lost in this process, and whether these were useful at all in the given context, or were correctly identified as spurious during the pruning procedure.

Bibliography

Adhikari, A., Ram, A., Tang, R., and Lin, J. (2019). DocBERT: BERT for document classification. *arXiv*, abs/1904.08398.

Adi, Y., Kermany, E., Belinkov, Y., Lavi, O., and Goldberg, Y. (2017). Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *ICLR*.

Ba, J. and Caruana, R. (2014). Do deep nets really need to be deep? In *NeurIPS*, pages 2654–2662.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *ICLR*.

Belinkov, Y. and Glass, J. R. (2019). Analysis methods in neural language processing: A survey. In Burstein, J., Doran, C., and Solorio, T., editors, *NAACL-HLT*, pages 3348–3354. ACL.

Bucila, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 535–541. ACM.

Cheong, R. and Daniel, R. (2019). transformers.zip: Compressing transformers with pruning and quantization. http://web.stanford.edu/class/cs224n/reports/custom/15763707.pdf.

Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. (2019). What does BERT look at? an analysis of BERT's attention. In *ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, pages 160–167. ACM.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. P. (2011). Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537.

Conneau, A. and Kiela, D. (2018). SentEval: An evaluation toolkit for universal sentence representations. In Calzolari, N., Choukri, K., Cieri, C., Declerck, T., Goggi, S., Hasida, K., Isahara, H., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., Piperidis, S., and Tokunaga, T., editors, *LREC*. European Language Resources Association.

Conneau, A., Kruszewski, G., Lample, G., Barrault, L., and Baroni, M. (2018). What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In *ACL*, pages 2126–2136. ACL.

Conneau, A. and Lample, G. (2019). Cross-lingual language model pretraining. In *NeurIPS*, pages 7057–7067.

Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. In *NeurIPS*, pages 3079–3087.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In *ACL*, pages 2978–2988. ACL.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186. ACL.

Dos Santos, C. and Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78.

Gieske, S. A. (2017). Inflecting verbs with word embeddings: A systematic investigation of morphological information captured by German verb embeddings (MSc thesis). https://esc.fnwi.uva.nl/thesis/centraal/files/f1156967543.pdf.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. (2018). Explaining explanations: An overview of interpretability of machine learning. *arXiv*, abs/1806.00069.

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv*, abs/1706.02677.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv*, abs/1308.0850.

Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding. In *ICLR*.

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

Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length and Helmholtz free energy. In *NeurIPS*, pages 3–10.

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

Huang, Z. and Wang, N. (2017). Like what you like: Knowledge distill via neuron selectivity transfer. *arXiv*, abs/1707.01219.

Huang, Z., Xu, P., Liang, D., Mishra, A., and Xiang, B. (2020). TRANS-BLSTM: transformer with bidirectional LSTM for language understanding. *arXiv*, abs/2003.07000.

Jawahar, G., Sagot, B., and Seddah, D. (2019). What does BERT learn about the structure of language? In *ACL*, pages 3651–3657. ACL.

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. *arXiv*, abs/1909.10351.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *EMNLP*, pages 1700–1709. ACL.

Kim, B., Wattenberg, M., Gilmer, J., Cai, C. J., Wexler, J., Viégas, F. B., and Sayres, R. (2018). Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In *ICML*, volume 80 of *PMLR*, pages 2668–2677. PMLR.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *EMNLP*, pages 1746–1751. ACL.

Kim, Y. and Rush, A. M. (2016). Sequence-level knowledge distillation. In *EMNLP*, pages 1317–1327. ACL.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR*.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *NeurIPS*, pages 3294–3302.

Kovaleva, O., Romanov, A., Rogers, A., and Rumshisky, A. (2019). Revealing the dark secrets of BERT. In *EMNLP-IJCNLP*, pages 4364–4373. ACL.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *NeurIPS*, pages 1106–1114.

Kusner, M., Sun, Y., Kolkin, N., and Weinberger, K. (2015). From word embeddings to document distances. In *ICML*, volume 37 of *PMLR*, pages 957–966.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. In *NAACL-HLT*, pages 260–270. ACL.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv*, abs/1909.11942.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C., and Kang, J. (2019). BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Lin, Y., Tan, Y. C., and Frank, R. (2019). Open sesame: Getting inside BERT's linguistic knowledge. In *ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 241–253. ACL.

Lin, Z., Feng, M., dos Santos, C. N., Yu, M., Xiang, B., Zhou, B., and Bengio, Y. (2017). A structured self-attentive sentence embedding. In *ICLR*.

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

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *ICLR*.

Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *EMNLP*, pages 1412–1421. ACL.

Mao, Y., Wang, Y., Wu, C., Zhang, C., Wang, Y., Yang, Y., Zhang, Q., Tong, Y., and Bai, J. (2020). LadaBERT: Lightweight adaptation of BERT through hybrid model compression. *arXiv*, abs/2004.04124.

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.

Michel, P., Levy, O., and Neubig, G. (2019). Are sixteen heads really better than one? In *NeurIPS*, pages 14014–14024.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *ICLR*.

Mikolov, T., Karafiát, M., Burget, L., Černocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048. ISCA.

Mirzadeh, S., Farajtabar, M., Li, A., and Ghasemzadeh, H. (2019). Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *arXiv*, abs/1902.03393.

Mukherjee, S. and Awadallah, A. H. (2019). Distilling transformers into simple neural networks with unlabeled transfer data. *arXiv*, abs/1910.01769.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Fürnkranz, J. and Joachims, T., editors, *ICML*, pages 807–814. Omnipress.

Papamakarios, G. (2015). Distilling model knowledge (MSc thesis). *arXiv*, abs/1510.02437v1.

Pires, T., Schlinger, E., and Garrette, D. (2019). How multilingual is multilingual BERT? In Korhonen, A., Traum, D. R., and Màrquez, L., editors, *ACL*, pages 4996–5001. ACL.

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*.

Rogers, A., Kovaleva, O., and Rumshisky, A. (2020). A primer in BERTology: What we know about how BERT works. *arXiv*, abs/2002.12327.

Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2015). FitNets: Hints for thin deep nets. In *ICLR*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1987). Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 318–362. MIT Press, Cambridge, MA, USA.

Sajjad, H., Dalvi, F., Durrani, N., and Nakov, P. (2020). Poor man's BERT: smaller and faster transformer models. *arXiv*, abs/2004.03844.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *NeurIPS EMC$\hat{2}$ Workshop*.

Sau, B. B. and Balasubramanian, V. N. (2016). Deep model compression: Distilling knowledge from noisy teachers. *arXiv*, abs/1610.09650.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *ACL*, pages 1715–1725. ACL.

Shi, X., Padhi, I., and Knight, K. (2016). Does string-based neural MT learn source syntax? In *EMNLP*, pages 1526–1534. ACL.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep inside convolutional networks: Visualising image classification models and saliency maps. In *ICLR*.

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 *EMNLP*, pages 1631–1642. ACL.

Strobelt, H., Gehrmann, S., Behrisch, M., Perer, A., Pfister, H., and Rush, A. M. (2019). Seq2seq-Vis: A visual debugging tool for sequence-to-sequence models. *IEEE Trans. Vis. Comput. Graph.*, 25(1):353–363.

Sucik, S. (2019). Pruning BERT to accelerate inference. https://blog.rasa.com/pruning-bert-to-accelerate-inference/.

Sun, C., Myers, A., Vondrick, C., Murphy, K., and Schmid, C. (2019a). VideoBERT: A joint model for video and language representation learning. In *IEEE-ICCV*, pages 7463–7472. IEEE.

Sun, S., Cheng, Y., Gan, Z., and Liu, J. (2019b). Patient knowledge distillation for BERT model compression. In *EMNLP-IJCNLP*, pages 4322–4331. ACL.

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

Tang, R., Lu, Y., and Lin, J. (2019a). Natural language generation for effective knowledge distillation. In *EMNLP-IJCNLP Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo)*, pages 202–208.

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

Tenney, I., Das, D., and Pavlick, E. (2019a). BERT rediscovers the classical NLP pipeline. In *ACL*, pages 4593–4601. ACL.

Tenney, I., Xia, P., Chen, B., Wang, A., Poliak, A., McCoy, R. T., Kim, N., Van Durme, B., Bowman, S., Das, D., et al. (2019b). What do you learn from context? probing for sentence structure in contextualized word representations. In *ICLR*.

Tsai, H., Riesa, J., Johnson, M., Arivazhagan, N., Li, X., and Archer, A. (2019). Small and practical BERT models for sequence labeling. In *EMNLP-IJCNLP*, pages 3630–3634. ACL.

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 *NeurIPS*, pages 5998–6008.

Wallace, E., Tuyls, J., Wang, J., Subramanian, S., Gardner, M., and Singh, S. (2019). AllenNLP Interpret: A framework for explaining predictions of NLP models. In *EMNLP-IJCNLP*, pages 7–12. ACL.

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. In *EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355. ACL.

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

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. *ArXiv*, abs/1910.03771.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., ÅĄukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv*, abs/1609.08144.

Yu, S., Kulkarni, N., Lee, H., and Kim, J. (2018). On-device neural language model based word prediction. In *COLING*, pages 128–131. ACL.

Zeiler, M. D. (2012). ADADELTA: An adaptive learning rate method. *arXiv*, 1212.5701.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *ECCV*, volume 8689, pages 818–833. Springer.

Appendix A

Datasets

intent name	description	example
`affirm`	affirmative response	`yes please!`
`ask_builder`	asking Sara who built her	`who developed you`
`ask_faq_channels`	asking Sara about the messaging channels that Rasa tools support	`what chat channels does rasa uses`
`ask_faq_community_size`	asking Sara about the size of the Rasa contributor community	`Is the~community large?`
`ask_faq_differencecorenlu`	asking Sara about the difference between two major components of the Rasa tools: Rasa NLU and Rasa Core	`what is the~difference between core and nlu?`
`ask_faq_languages`	asking Sara about the languages supported by Rasa tools	`do you support french ?`
`ask_faq_opensource`	asking Sara if Rasa products are open source	`are you full open source`
`ask_faq_platform`	asking Sara about the Rasa Platform product	`tell me what is platform`
`ask_faq_python_version`	asking Sara about the version of Python supported by Rasa tools	`which python version`
`ask_faq_slots`	asking Sara about *slots*, a concept in Rasa tools for holding human-provided contextual information during conversations	`what do you mean by slots?`

Continued on next page

Table A.1 – *Continued from previous page*

ask_faq_tutorials	asking Sara about tutorials on using Rasa tools	is there a~tutorial for this?
ask_faq_voice	asking Sara about the possibility to create a voice assistant using Rasa	you have speech recognition?
ask_faq_what_is_forum	asking Sara about the online Rasa community forum	what can I~post in the~forum?
ask_how_contribute	asking Sara how one can contribute to the Rasa open-source project	How can I~add code to Rasa
ask_howbuilt	asking Sara how she was built	so how were you made?
ask_howdoing	asking Sara how she is doing	Hows it going
ask_howold	asking Sara about her age	do you know how old you are?
ask_isbot	asking Sara if she is a bot	are you really a~bot
ask_languagesbot	asking Sara about the languages she can speak	how many languages are you fluent in?
ask_question_in_forum	asking Sara a question about the Rasa community forum	how can I~leave a~query in the~forum?
ask_restaurant	asking Sara to recommend a restaurant	Where should I~eat?
ask_time	asking Sara about the time	tell me the~time it is.
ask_weather	asking Sara about the weather	excellent - is it hot in Berlin?
ask_whatismyname	asking Sara to tell the person's name	can you tell me my name?
ask_whatisrasa	asking Sara what Rasa is	OK can u brief me Abt rasa
ask_whatspossible	asking Sara about the things she can do/help with	how u can help me

Continued on next page

Table A.1 – *Continued from previous page*		
ask_when_next_event	asking Sara about the next scheduled Rasa community event	what date is the~next community event?
ask_wherefrom	asking Sara where she is from	where did you grow up?
ask_which_events	asking Sara about the current Rasa community events	what sort of social events are we throwing?
ask_whoami	asking Sara who the human person is	tell me who I~am?
ask_whoisit	asking who is it (like on the phone)	who am i talking to
ask_why_contribute	asking Sara about the reasons to contribute to Rasa	Why should I~contribute to your code?
bye	ending a conversation with Sara by saying bye	take care
canthelp	telling Sara she cannot help with what is needed	i guess you can't help me then
contact_sales	asking Sara about ways to contact the Rasa sales team	i want to talk to sales
deny	provide a negative, denying response to Sara	no sorry
enter_data	providing information asked for by Sara	the~assistant is in dutch, or my name is __PERSON_NAME__
greet	saying hi to Sara	hey let's talk
handleinsult	telling an insult to Sara	i hate your dumb face
how_to_get_started	asking Sara how one can get started with Rasa tools	how to build a~chatbot
human_handoff	asking to be put through to a human instead of the Sara bot	let me speak with a~real person please
install_rasa	asking Sara about installing Rasa	i need help setting up

Continued on next page

Table A.1 – *Continued from previous page*

`next_step`	asking Sara to proceed to the next step	`next step please`
`nicetomeeyou`	saying to Sara it is nice to meet her	`Good to meet you!`
`nlu_generation_tool_recommendation`	asking Sara about tools that can be used to generate more NLU training data (intent examples like these)	`i need more nlu data`
`nlu_info`	asking Sara about the Rasa NLU tool	`what is a~intent?`
`out_of_scope`	an out-of-scope message not falling into any of the other intent categories	`how to climb the~tree?`
`pipeline_recommendation`	asking Sara about the pipeline configuration used when building bots using Rasa tools	`what pipeline should i use?`
`rasa_cost`	asking Sara about the price of Rasa products	`is rasa core paid?`
`react_negative`	negative reaction (typically in response to Sara asking how the person is feeling)	`so sad :(`
`react_positive`	positive reaction (typically in response to Sara asking how the person is feeling)	`you are cool man`
`signup_newsletter`	asking Sara about signing up for a newsletter	`i want on that dope newsletter`
`source_code`	asking Sara about her source code	`your code please`
`switch`	asking Sara about switching from a competitor tool to Rasa	`How to migrate from DialogFlow to Rasa?`
`technical_question`	asking Sara an assorted technical questions	`do you have docker image for rasa?`
`telljoke`	asking Sara to tell a jok	`say a~funny joke`
`thank`	thanking Sara	`amazing, thanks`

Table A.1: A complete list of the intents found in the Sara dataset.

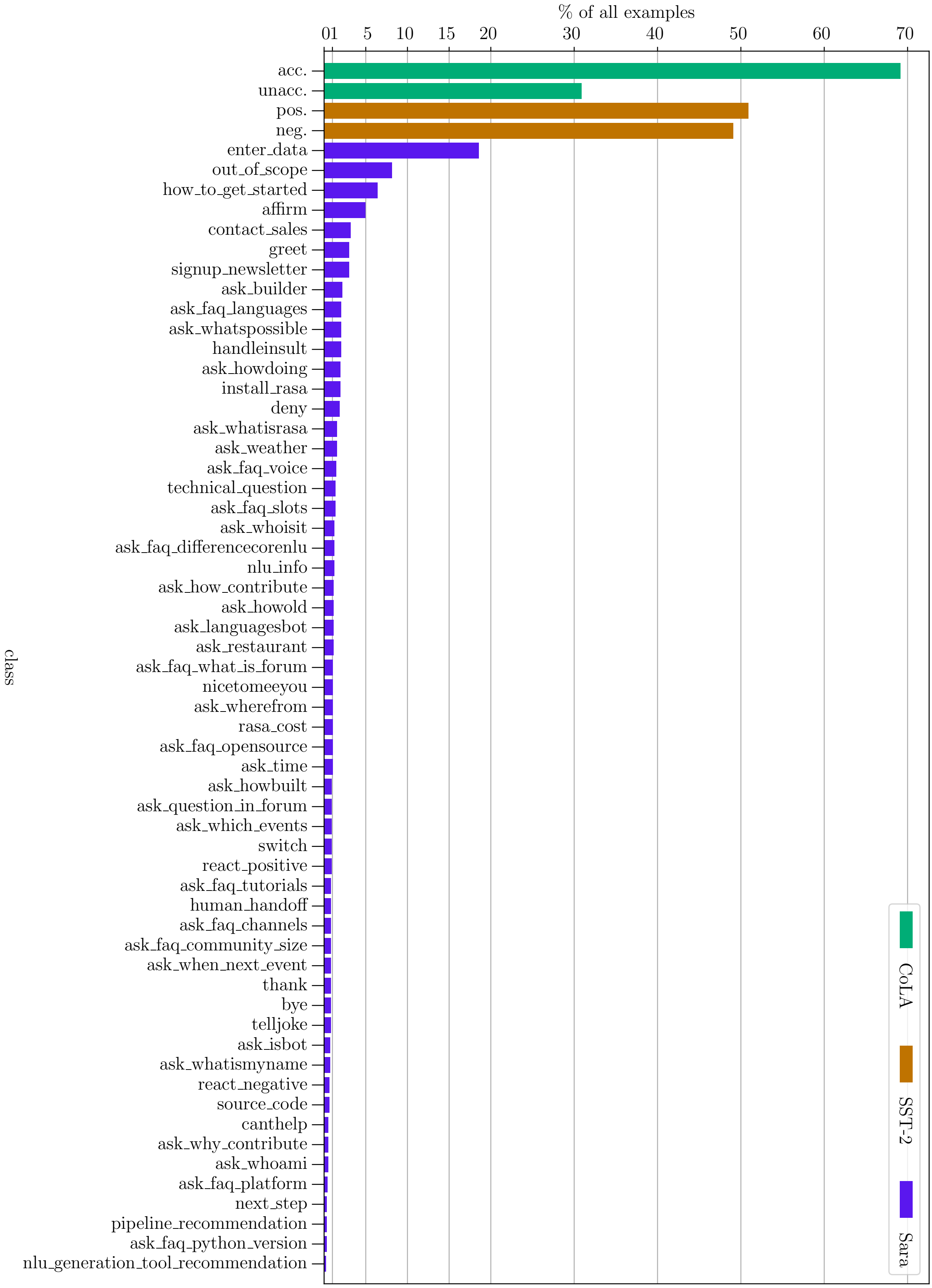


Figure A.1: Class distribution in the different downstream task datasets; not that this distribution is roughly the same across the different portions of each dataset.

Appendix B

Student hyperparameter exploration

B.1 Initial exploration on CoLA

B.1.1 Choosing learning algorithm and learning rate

Because Tang et al. (2019a) report not tuning their BiLSTM hyperparameters, I verify their choices. In particular, the use of the Adam learning algorithm is explored – a widely used and improved version of the Adadelta algorithm which is used originally.

For both students, I try a wide range of η values with Adam: 5×10^{-3}, 1.5×10^{-3}, 5×10^{-4}, 1.5×10^{-4}, 5×10^{-5}, 1.5×10^{-5}, 5×10^{-6}.

Fig. B.1 shows that for all students the ideal η is around 5×10^{-4}. Much larger and much smaller values leading to poor learning, in particular the largest $\eta=5\times10^{-3}$ "kills" the learning of BERT_S entirely due to gradient explosion. As expected, BERT_S initialised from scratch performs worse than when initialised from the wordpiece embeddings of BERT_T. However, the differences are not large.

As discussed previously, BERT_S converges much slower than LSTM_S, hence the 30-epoch and 60-epoch training budgets for LSTM_S and BERT_S, respectively. Additionally, it is apparent that LSTM_S performs significantly better than BERT_S even though the sizes of the models are comparable.

In the case of LSTM_S, the Adadelta algorithm is outperformed by Adam. From now onwards, I use Adam with both students, in all cases with $\eta=5\times10^{-4}$.

B.1.2 Choosing learning rate scheduling and batch size

Tang et al. (2019b) use no learning rate scheduling, but they report small batch sizes ($B=50$) to work better than the usual, larger batches. Hence, for LSTM_S I first verify their claims and subsequently move on to η scheduling. For BERT_S, inspired by Sanh et al. (2019) who take advantage of scheduling (both in terms of warmup and decay), I explore η scheduling first (as a continuation from exploring η values), and then look at various B values.

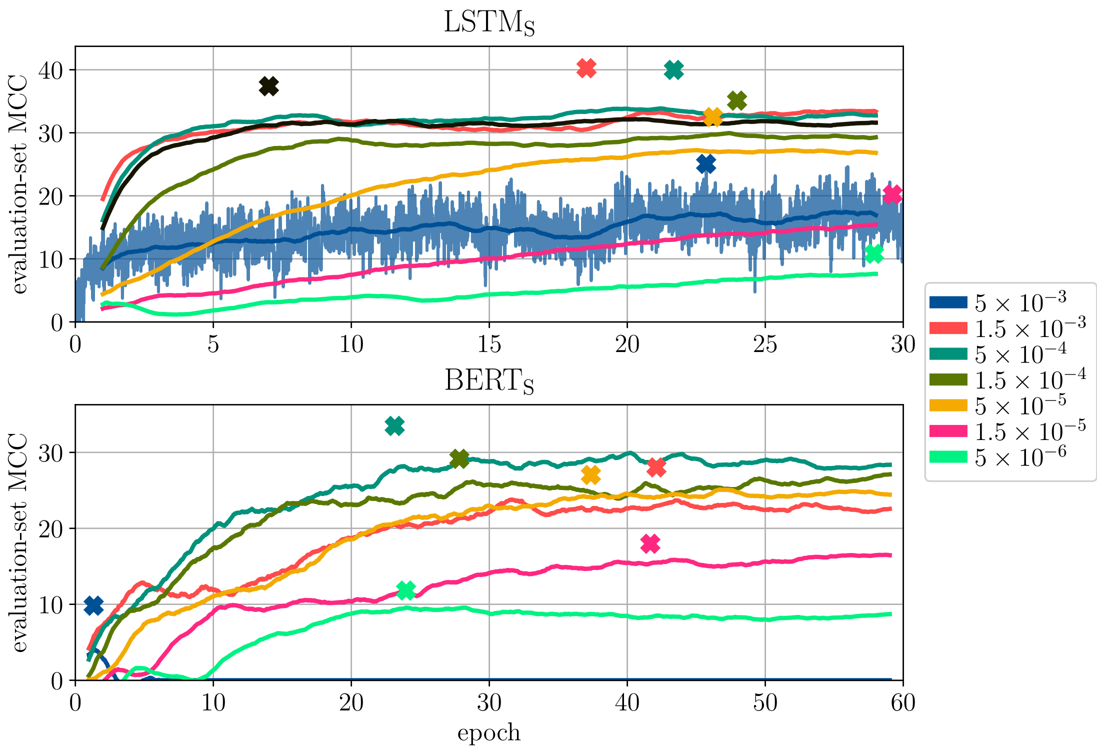


Figure B.1: Comparing various η values on CoLA. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window. In particular, notice that the best-score points (crosses) are well above the smoothed lines – this is because the (unsmoothed) evaluation score varies a lot, as illustrated in the upper plot for LSTM$_S$ with $\eta = 5 \times 10^{-3}$.

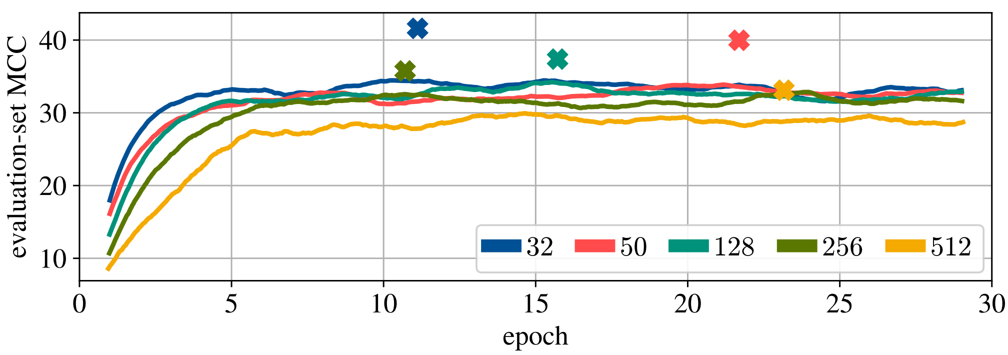


Figure B.2: Comparing various batch sizes for LSTM$_S$ on CoLA. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

As Fig. B.2 shows, LSTM$_S$ clearly does prefer small batch sizes. However, training with tiny minibatches takes very long – compare ~25min for $B = 512$ with ~7h for $B = 32$. Hence, I restrain from trying even smaller batch sizes and use $B = 32$ for LSTM$_S$ in all further experiments.

Fig. B.3 shows the results of exploring various warmup durations for both students. Note that for LSTM$_S$, whose training budget is only 30 epochs, I did not try the long warmup duration of 20 epochs, only up to 15 epochs.

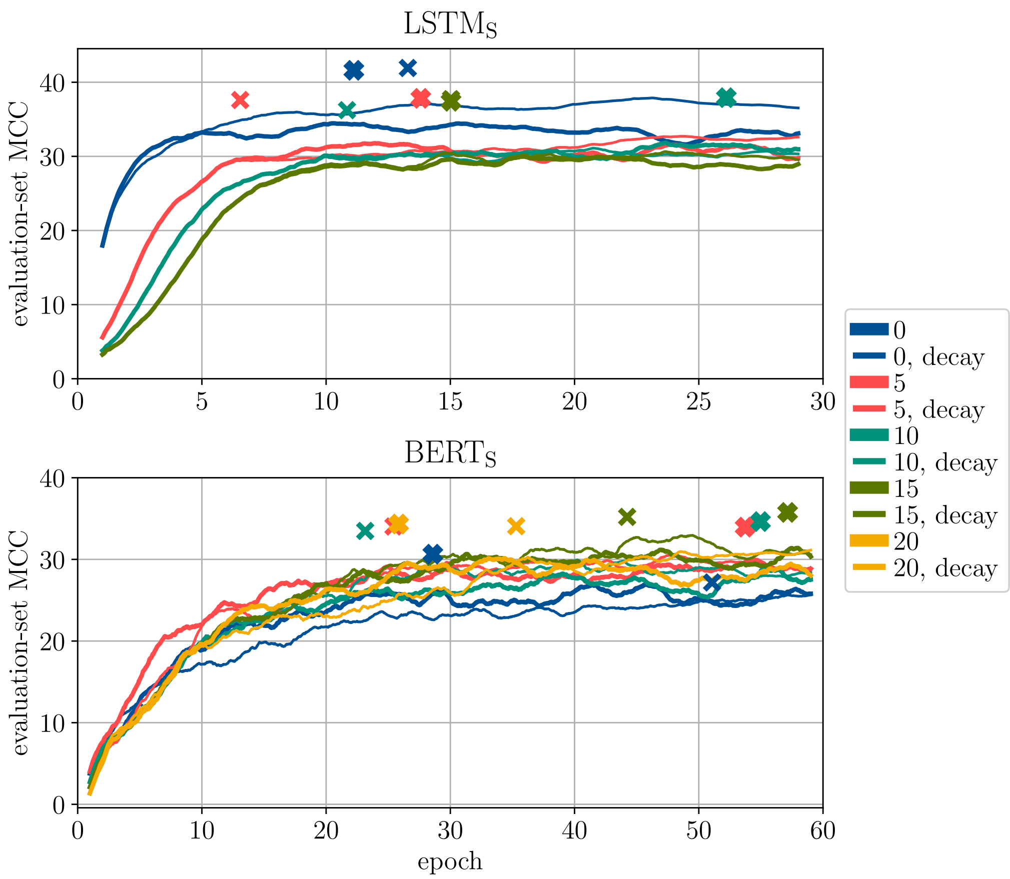


Figure B.3: Comparing warmup durations E_w on CoLA, with the optional learning rate decay to 0 over the remaining training epochs. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

In the case of LSTM$_S$, Fig. B.3 shows that the longer the warmup duration, the slower the model converges. This is understandable because during warmup, learning happens less aggressively – and hence more slowly – due to the smaller learning rate. More importantly, the graph shows that η decay does not significantly affect training, but it can help to prevent the model from overfitting the training data. This is most visible for $E_w = 0$ where LSTM$_S$'s performance starts to slowly decrease after 20 epochs in the absence of η decay. All in all, using the full η from the beginning of training is the best option, and η decay can only improve things. $E_w = 0$ with decay is used for LSTM$_S$ in all further experiments.

In the case of BERT$_S$, the only clear result visible from Fig. B.3 is that BERT$_S$ performs poorly without η warmup. For non-zero warmup durations, there are no significant differences in the best-performance points (marked by crosses) or in the convergence speed. In all further experiments with BERT$_S$, I use $E_w = 15$ and η decay – the configuration which shows the highest stable performance level in Fig. B.3 in later epochs (beyond epoch 35).

The batch size exploration in Fig. B.4 shows that BERT$_S$ performs best with mid-sized batches of 128-256 examples. Too large batch size ($B = 512$) as well as very small batches of 32-64 make BERT$_S$ underperform (with tiny batches of 32 being particularly detrimental). In all further experiments, $B = 128$ is used with BERT$_S$.

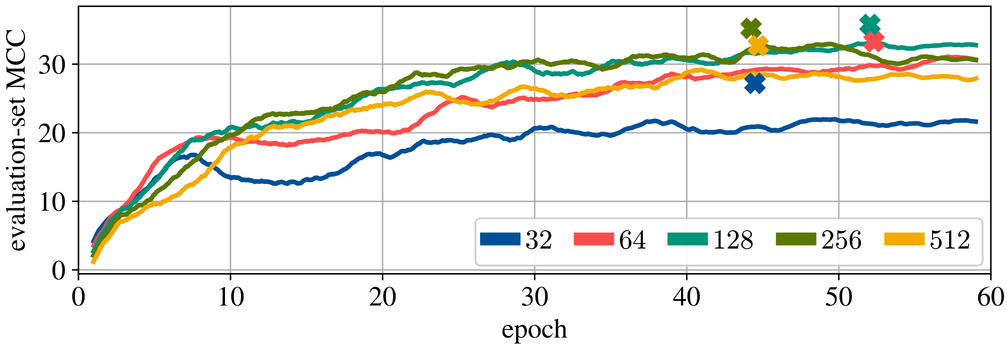


Figure B.4: Comparing various batch sizes for BERT$_S$ on CoLA. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

B.2 Optimising students for each downstream task

In this section, I provide details of explore different ways of initialising student models with language knowledge, and the effect of model size, separately for each downstream task.

B.2.1 Choosing embedding type and mode

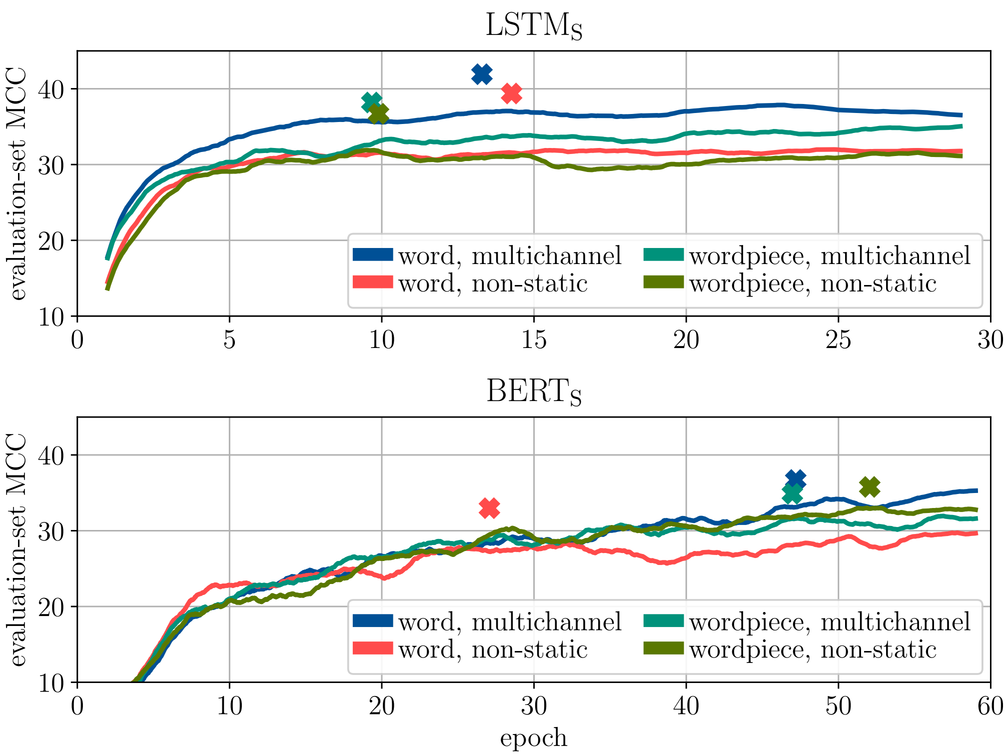


Figure B.5: Comparing embedding types and modes on CoLA. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

Fig. B.5 shows how different type and mode combinations affect knowledge distillation on CoLA. With LSTM_S, the multichannel mode is preferred to non-static and word2vec embeddings are preferred to wordpieces; hence, I use the multichannel mode with word2vec in further experiments (same as Tang et al. (2019a)). For BERT_S, the differences are smaller, yet it is clear that word-level embeddings benefit from using the frozen and unfrozen versions provided by the multichannel mode. In further experiments, the multichannel mode combined with word-level word2vec embeddings is used.

For SST-2, I only compare the best word-level and the best wordpiece-level combination for each model as observed from Fig. B.5. The results are shown in Fig. B.6. Notice the scale of the y axis: In particular, the students perform roughly the same (unlike on CoLA) and any relative differences observed in Fig. B.6 are much smaller than the differences observed on CoLA in Fig. B.5. Hence, I refrain from making conclusions about which embedding type and mode works better; I merely choose to use the word-level embeddings with the multichannel mode in all further experiments on SST-2 (my decision is based on the best evaluation scores marked by crosses in Fig. B.6).

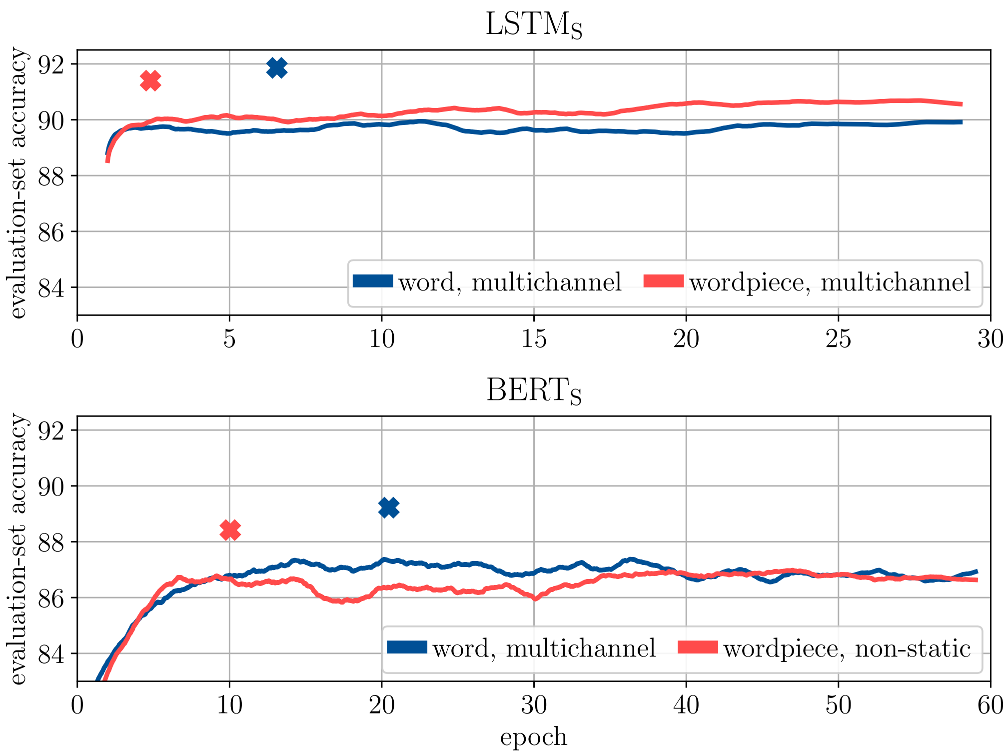


Figure B.6: Comparing embedding types and modes on SST-2. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

Results on Sara are shown in Fig. B.7. Here again, the relative differences in performance are small, but using wordpieces helps both students converge faster and reach slightly better performance levels. This can be a result of the word2vec vocabulary not capturing well the conversational language in Sara examples, for instance the utterance “yesyesyes” would be treated simply as one out-of-vocabulary word, whereas wordpieces have the potential to encode it as the word “yes” repeated 3x. In all further experiments on Sara I use the wordpiece embeddings, using the multichannel mode in LSTM_S and the non-static mode in BERT_S.

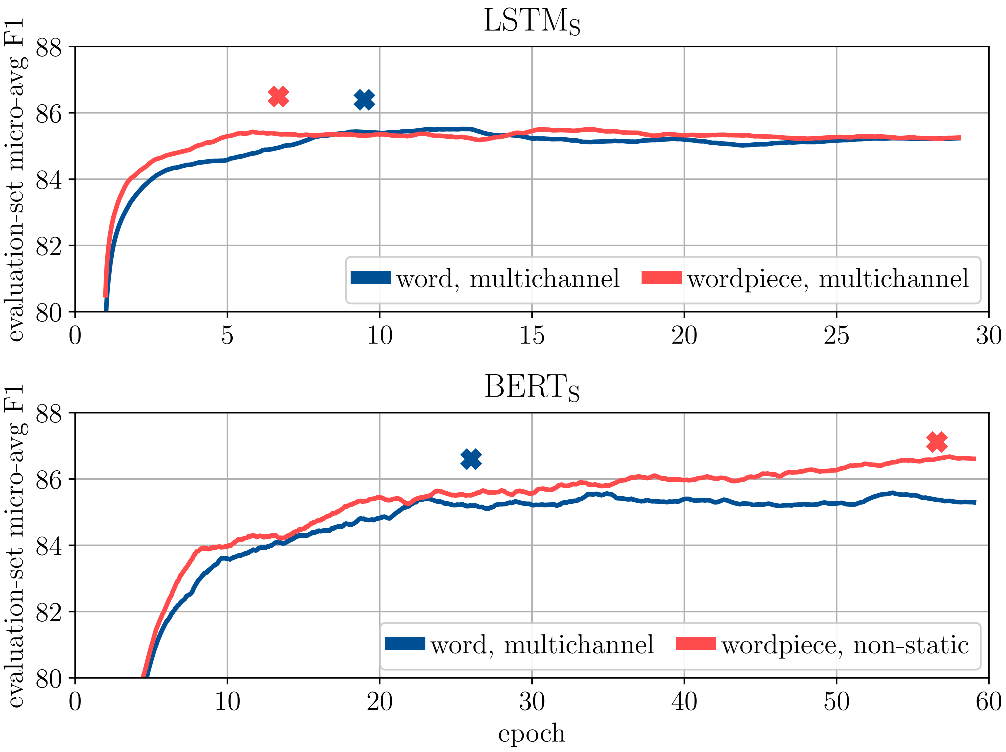


Figure B.7: Comparing embedding types and modes on Sara. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

All in all, the multichannel mode seems to be generally superior to the non-static mode which lacks the frozen version of embeddings. Initialisation from the word-level word2vec embeddings also works better than wordpieces where examples tend to contain legitimate English (CoLA and SST-2). As for the performance gap between LSTM_S and BERT_S, I conclude that it cannot be explained by differences in embedding type/mode, and is most likely a consequence of different student architectures.

While wordpiece embeddings have the advantage of being fine-tuned on the particular downstream task (as part of teacher fine-tuning), word2vec contains more general knowledge stored at the word level. Importantly, the wordpiece vocabulary contains the most frequent words in their entirety; only the less frequent ones are split into pieces. Thus, for frequent words, their word2vec and wordpiece embeddings will differ only in the way they are trained. Naturally, since the wordpiece vocabulary has only 30,522 tokens while word2vec has 3,000,000, there are many words covered by word2vec for which the wordpiece embeddings have to be assembled from multiple pieces. On these words – frequent enough to be in the word2vec vocabulary, but not the most frequent ones – word2vec could have an advantage. Once we move beyond the words covered by word2vec to rare tokens like "yesyesyes", wordpieces become the preferred approach. Clearly, no one approach is generally the best, and the decision should ideally be made individually for each downstream task.

B.2.2 Choosing student size

As the last parameter, I explore the size of each student. In particular, I try to reduce the performance gap on CoLA between BERT_T and both students by making the students larger. On SST-2 and Sara, the 2.4-million-parameter students already achieve scores very close to those of the teacher, and I refrain from exploring larger students – instead, I explore smaller student sizes.

There are two main ways of increasing the student size: By increasing the model "width" and the "depth". By width, I mean the dimensionality of the model's layers and internal representations. Depth means adding more layers. While larger width allows the models to extract and maintain richer token representations, adding layers adds more steps to the models' processing pipelines, allowing for more abstract and task-specific representations to be extracted in the end.

In BERT_S, I manipulate width by increasing by a set factor W the hidden dimensionality d_h, the intermediate dimensionality d_I, and the number of self-attentional heads A. Depth is manipulated by changing the number of encoder layers L by the factor D. In LSTM_S, I change model width by scaling by W the LSTM layer width d_{LSTM} and the fully-connected layer width d_{FC}, which are originally set to 300 and 400, respectively. Depth is changed by increasing the number of LSTM layers (originally just one) by the factor D. The concrete dimensions of up- and down-scaled students are shown in Tab. B.1 (BERT_S) and in Tab. B.2 (LSTM_S).

W	d_h	d_I	A
1/16	13	47	1
1/8	26	94	1
1/4	51	188	1
1/3	68	250	1
1/2	102	375	2
1	204	750	3
2	408	1500	6
3	612	2250	9
4	816	3000	12

D	L
1/4	1
1/3	2
1/2	3
1	5
2	10
3	15

Table B.1: BERT_S dimensions for different width scalings (left) and depth scalings (right). The default size with 2.4 million parameters corresponds to $W = 1$, $D = 1$.

B.2.2.1 CoLA

With the teacher's evaluation-set MCC of 59.9 being much higher than the student performance observed so far (around 40), I up-scale both students, aiming for 90% of the teacher performance while keeping the student size smaller than the 340-million-parameter teacher.

As observed in preliminary experiments with large BERT_S versions, their learning suffers from gradient explosion due to the learning rate being too large for the models. For an

W	d_{LSTM}	d_{FC}
1/32	9	13
1/16	19	25
1/8	37	50
1/4	75	100
1/2	150	200
1	300	400
2	600	800
3	900	1200
4	1200	1600
5	1500	2000

D	L
1	1
2	2
3	3
4	4
5	5

Table B.2: LSTM$_S$ dimensions for different width scalings (left) and depth scalings (right). The default size with 2.4 million parameters corresponds to $W = 1$, $D = 1$.

example, see Fig. B.8 where the gradient explosion happens around epoch 7 and the model score (MCC) then falls to 0 and stays there.

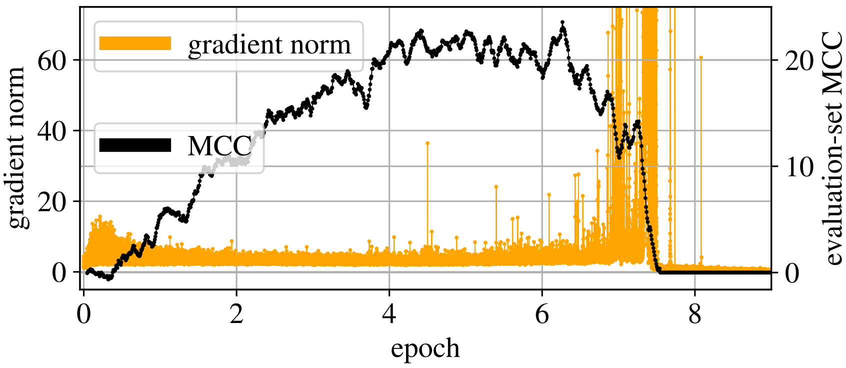


Figure B.8: Gradient explosion in BERT$_S$ with $W = 2$ and $D = 2$. The MCC values have been smoothed with a 0.1-epoch-wide sliding average window.

Without further extensive exploration of optimal learning rate values for each BERT$_S$ size[1], I choose better learning rate values manually. Because of the use of η warmup, I can monitor the learning progress for varying η values in the early training epochs, as shown in Fig. B.9. I approximately identify the point in training beyond which the learning slows down (and later degrades altogether) due to large gradients. This way, I approximately identify the largest learning rate that still leads to learning, not to gradient explosion. In the concrete example in Fig. B.9, I choose the point in training after 2.5 epochs, where the learning rate is approximately $\eta = 8 \times 10^{-5}$, and use this value with the concrete model size.

With the new learning rates manually estimated individually for each student size, none of the larger versions of BERT$_S$ experiences gradient explosion. The LSTM students all use the same learning rate as this does not lead to any issues. Fig. B.10 presents the

[1] This would be extremely time-consuming because the larger versions take over 3 days to train.

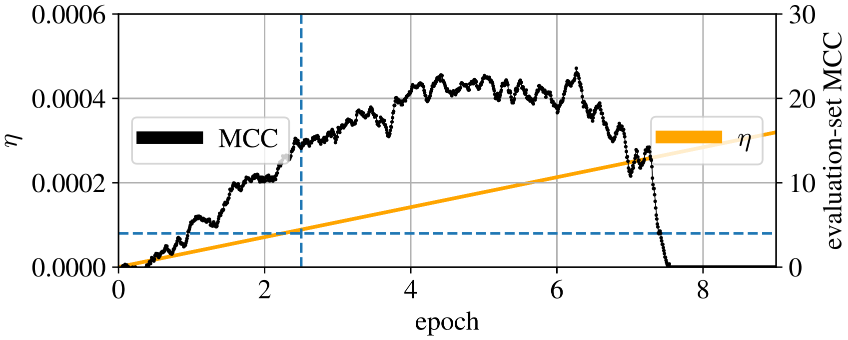
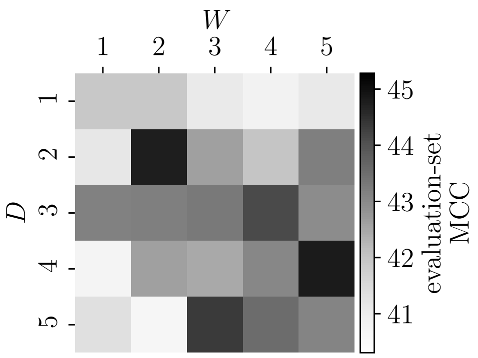
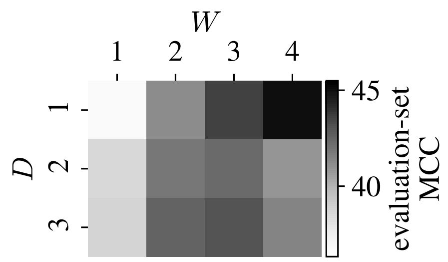


Figure B.9: Learning progress (MCC over time) vs η for BERT$_S$ with $W = 2$ and $D = 2$ – the model experiencing gradient explosion in Fig. B.8. The dashed lines show the position I identify as the approximate latest point of training before the learning starts to slow down, and the learning rate at that position. The MCC values have been smoothed with a 0.1-epoch-wide sliding average window.



(a) Different sizes of LSTM$_S$.



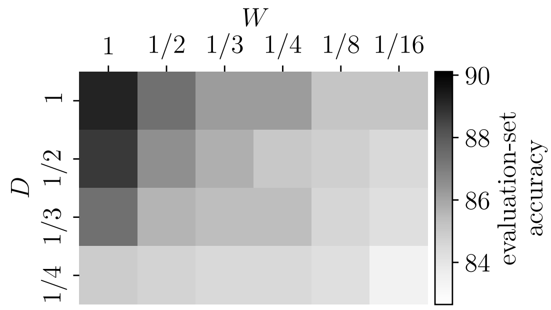
(b) Different sizes of BERT$_S$.

Figure B.10: Best evaluation-set performance for the different student sizes on CoLA.

results. While some larger students outperform the original, 2.4-million-parameter ones, the trends are not consistent. For LSTM$_S$ in particular, there is no clear correlation between student width or depth and the performance. For BERT$_S$, which starts as a relatively deep model with 5 layers, making it wider rather than deeper is helpful. For LSTM$_S$ which originally has only 1 hidden LSTM layer, increasing both the width and the depth can lead to better-performing models. Overall, both student architectures reach the best evaluation score of ~45, far below the teacher performance level of 59.9.

I refrain from exploring students even larger as the biggest students are already approching the teacher size: LSTM$_S$ with $W = 5$, $D = 5$ has ~247 million parameters and takes over 60h to train, and BERT$_S$ with $W = 4$, $D = 3$ has ~114 million parameters and takes over 6 days to train. As the best-performing students, I establish BERT$_S$ with $W = 4$, $D = 1$, and LSTM$_S$ with $W = 2$, $D = 2$.

(a) Different sizes of LSTM_S.



(b) Different sizes of BERT_S.

Figure B.11: Best evaluation-set performance for the different student sizes on SST-2.

B.2.2.2 SST-2

As previously observed, on SST-2, even the default 2.4-million-parameter students perform on par with the teacher. With no reason to try larger student sizes, I limit myself to exploring smaller student architectures, with the aim of keeping student accuracy above 90% of the teacher's score. With BERT_T achieving 91.5% accuracy, the 90% lower bound is at ~82% accuracy.

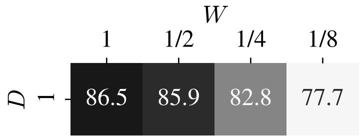
Fig. B.11 shows that accuracy stays high even for very small students. The smallest tried LSTM student ($W = 1/64$, 24,360 non-embedding parameters, ~14,000x smaller than BERT_T) still achieves 89.1% accuracy (~97% of the teacher's performance). The smallest tried BERT student ($W = 1/16$, $D = 1/4$, 2272 non-embedding parameters, ~150,000x smaller than BERT_T) achieves 83.5% accuracy (~91% of BERT_T's performance). What these results mean is that the SST-2 task is relatively easy. For good accuracy levels, a very minimalistic classifier is sufficient on top of the pre-trained embeddings – the representations obtained simply by encoding each word using word2vec already contain most of the knowledge needed to make good sentiment predictions.

Another insight from Fig. B.11b is that making BERT_S shallower affects the performance much less than making it slimmer. In other words, the 2.4-million-parameter BERT_S may be unnecessarily deep for the task, but it is not unnecessarily wide.

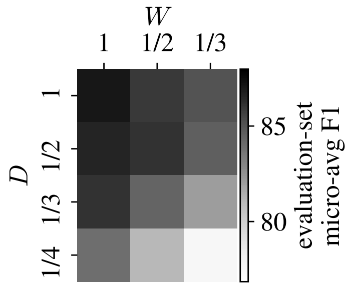
B.2.2.3 Sara

Similar to SST-2, Sara is an easy task. With BERT_T achieving $F1_{micro} = 87.5$, the 2.4-million-parameter BERT_S and LSTM_S already achieve 87.1 and 86.5, respectively. I further down-scale the students, see Fig. B.12. Similar to the results on SST-2, both students can be made much smaller while achieving over 90% of BERT_T's performance. The second smallest tried LSTM_S ($W = 1/4$) is 262x smaller than the teacher while retaining almost 95% of its performance. The BERT_S with $W = 1/2$ and $D = 1/4$, being 2500x smaller than BERT_T, retains ~93% of its performance.

Also similarly to SST-2, making BERT_S shallower has much weaker effect on the performance than making it thinner. In other words, the Sara task does not require very deep



(a) Different sizes of $\mathrm{LSTM_S}$.



(b) Different sizes of $\mathrm{BERT_S}$.

Figure B.12: Best evaluation-set performance for the different student sizes on Sara.

models, and keeping the representation dimensionality above certain level (in this case around 128 or above, corresponding to $W \geq 1/2$) is more important.

Appendix C

Details of model analysis

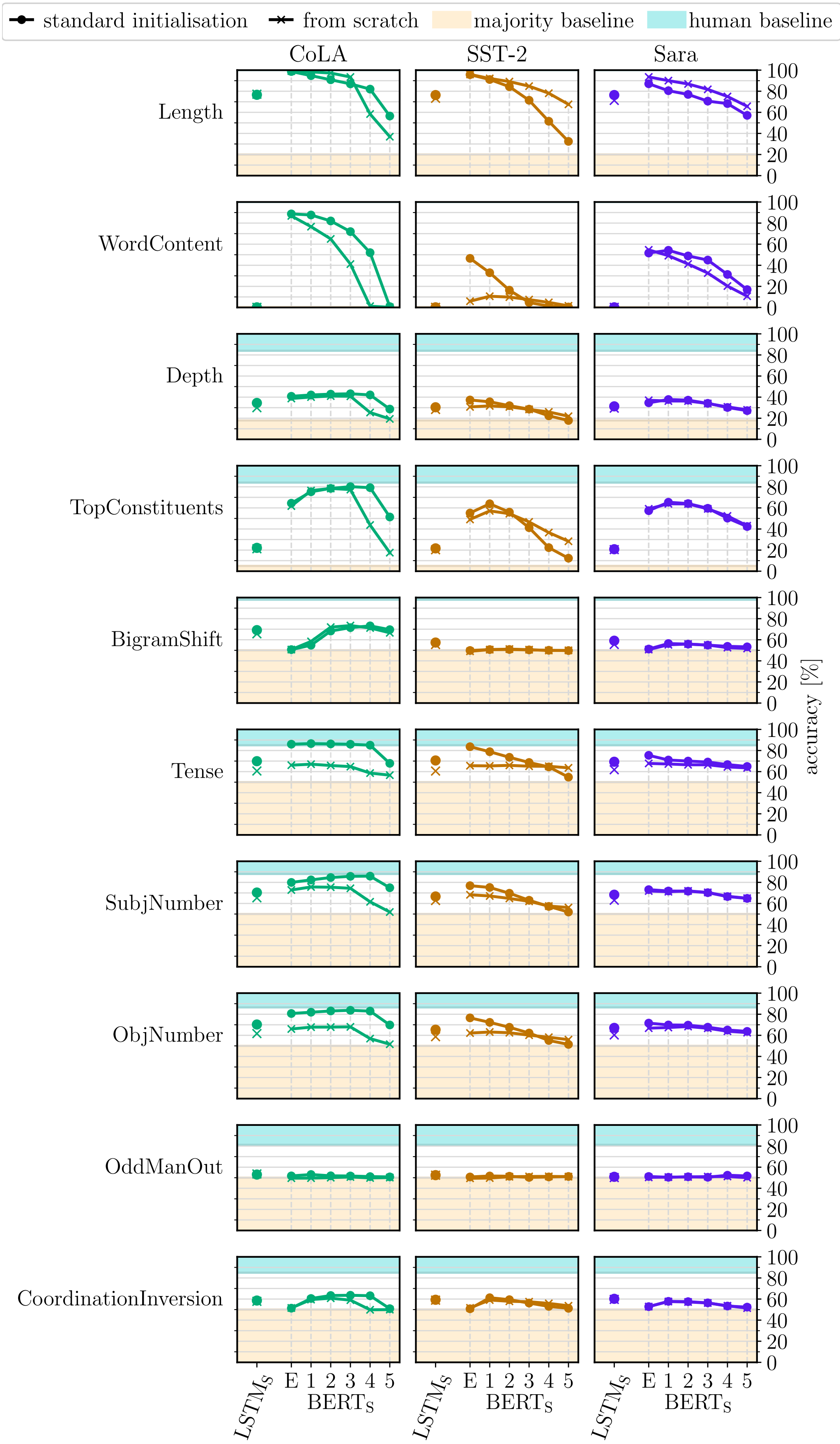


Figure C.1: Probing results comparing students initialised in the standard way (with embeddings from word2vec) with students initialised randomly and trained from scratch. Bounding the expected model performance from below and from above are again the majority-class baseline and the human performance baseline, as reported by Conneau et al. (2018).

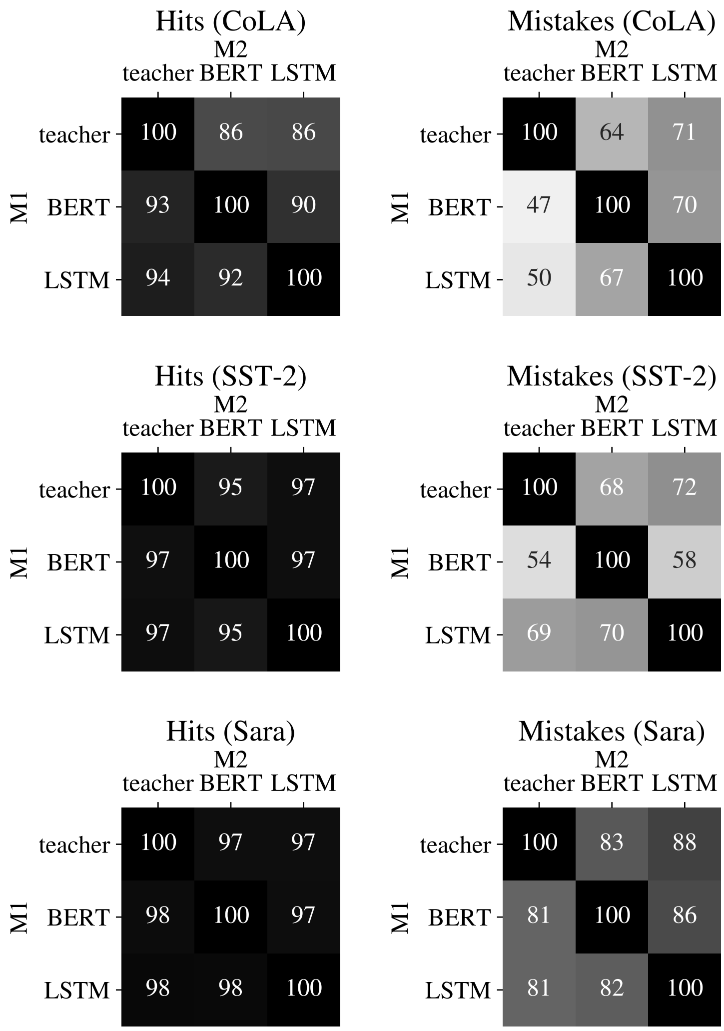


Figure C.2: Overlap of models' evaluation-set mistakes and hits. Each cell shows the fraction of hits or mistakes shared by models M1 and M2, as a percentage of the total hits or mistakes made by model M1.

L	O	sentence
0	o	i gave pete the book to impress.
1		i gave pete the book to impress him.
1	o	most of the fruit is rotten.
1		most of the fruit is ripened.
1		most of the fruit is spoiled.
1	o	the administration has issued a statement that it is willing to meet a student group, but i'm not sure which one.
1		the administration has issued a statement that it is willing to meet a student group.
1	o	the correspondence school sent bill a good typist.
1		the school sent bill a good typist.
1		the correspondence school sent bill a good baker.
1	o	the report that crime was declining surprised many people.
1		the report that crime was declining surprised many.
1		the report surprised many people.
0	o	the storm arrived while the picnic.
0		the storm arrived while the performance.
0		the storm arrived while the picnic was.
1		the storm arrived while the picnic was starting.
1		the storm arrived during the picnic.
1	o	they chased the man with the car.
1		they chased the man with a car.
1	o	they were going to meet sometime on sunday, but the faculty didn't know when.
1		they were going to meet sometime on sunday, but the faculty didn't know.
1		they were going to meet sometime on sunday.
0	o	we talked to them about there.
1		we talked to them about it.
1	o	bill's wine from france and ted's from california cannot be compared.
1		bill's wine from france and ted's wine from california cannot be compared.
1		bill's wine from france and ted's wine cannot be compared.
1		bill's wine and ted's wine cannot be compared.
0	o	brian threw the fence with the stick.
1		brian hit the fence with the stick.
1		brian attacked the fence with the stick.
1		brian poked the fence with the stick.
1		brian struck the fence with the stick.
0	o	chris was handed sandy a note.
1		chris handed sandy a note.
1		chris was handed a note.
0	o	dana walking and leslie ran.
1		dana walked and leslie ran.
0	o	john was struck as sick.

1		john was treated as sick.
1		john was seen as sick.
0	o	john whispered mary left.
1		john whispered that mary left.
1		john thought mary left.
1		john knew mary left.
0	o	leslie told us about us.
1		leslie told us about them.
1		leslie told them about us.
0	o	my heart is pounding me.
1		my heart is pounding.
0		my heart is beating me.
1		my heart is beating.
0	o	the children are fond that they have ice cream.
0		the children are eager that they have ice cream.
1		the children are fond of having ice cream.
1		the children are glad that they have ice cream.
1		the children are happy that they have ice cream.
0	o	the magazines were sent to herself by mary.
1		the magazines were sent by mary to herself.
1		the magazines were sent to her by mary.
0	o	the table was wiped by john clean.
1		the table was wiped clean by john.
1		the table was wiped by john.
0	o	we believed john to be a fountain in the park.
1		we believed there to be a fountain in the park.
0		we believed john to be a fountain.
0		we believed john to be a bench in the park.
0		we believed john to be a tree in the park.
1		we believed john to be a musician in the park.

Table C.1: Selected CoLA evaluation-set sentences classified correctly by the teacher model but incorrectly by one or both students. L = label (0 = unacceptable, 1 = acceptable), O = original sentence or a manually perturbed (and labelled) variant.