



Teacher-student knowledge distillation from BERT

Sam Sučík

MInf Project (Part 2) Report

Master of Informatics

School of Informatics

University of Edinburgh

2020

Abstract

TO-DO

Acknowledgements

I thank Steve Renals of University of Edinburgh and Vova Vlasov of Rasa for supervising me throughout the academic year; patiently listening to my never-ending reports and providing helpful and optimistic comments.

Many thanks also to Ralph Tang whose work inspired this project, and to Slávka Heželyová who constantly supported me and motivated me to explain all of my work in non-technical stories and metaphors.

Table of Contents

1	Introduction	7
1.1	Motivation	7
1.2	Aims	7
1.3	Contributions	8
2	Background [REVIEWED]	9
2.1	NLP before Transformers	9
2.2	Transformer-based NLP	12
2.2.1	Transformers	12
2.2.2	BERT	15
2.2.3	Newer and larger Transformer models	19
2.3	Teacher-student knowledge distillation	19
2.3.1	A brief introduction to knowledge distillation	19
2.3.2	Knowledge distillation in NLP	21
2.4	Interpreting NLP models	22
2.5	Conclusions [READY FOR REVIEW]	23
3	Datasets [REVIEWED]	25
3.1	Downstream tasks	25
3.1.1	Corpus of Linguistic Acceptability	26
3.1.2	Stanford Sentiment Treebank	26
3.1.3	Sara [READY FOR REVIEW]	27
3.2	Data augmentation for larger transfer datasets	29
3.3	Probing tasks	30
3.4	Conclusions [READY FOR REVIEW]	32
4	Methods and Implementation [REVIEWED]	33
4.1	Methods and objectives	33
4.2	System overview and adapted implementations	34
4.3	Implementation details	35
4.3.1	Teacher fine-tuning	35
4.3.2	Augmentation with GPT-2	36
4.3.3	BiLSTM student model	36
4.3.4	BERT student model	38
4.3.5	Knowledge distillation	38
4.3.6	Probing	40

4.4	Computing environment and runtimes	41
4.5	Conclusions [READY FOR REVIEW]	42
5	Training student models [READY FOR REVIEW]	43
5.1	General hyperparameter exploration on CoLA	43
5.1.1	Choosing learning algorithm and learning rate	44
5.1.2	Choosing learning rate scheduling and batch size	45
5.2	Optimising students for each downstream task	47
5.2.1	Choosing embedding type and mode	47
5.2.2	Choosing student size	50
5.3	Student training: conclusions	54
6	Analysing the models	57
6.1	Probing [READY FOR REVIEW]	57
6.1.1	Probing teacher models	58
6.1.2	Probing student models	59
6.1.3	What can students learn from scratch	62
6.2	Analysing the models' predictions	63
7	Overall discussion and conclusions	65
8	Future work	67
	Bibliography	69
A	Sara dataset	75
B	Probing	81

Chapter 1

Introduction

1.1 Motivation

- After the deep learning hype started, NLP went through an era of LSTMs. Since 2017, the area has been becoming dominated by Transformer models pre-trained on large unlabelled corpora.
- As newer and bigger Transformer-based models were proposed in 2018 and 2019, improving on the SOTA, it was becoming clearer that their big size and low speed was rendering them difficult to use (both train and deploy) in practice outside of research labs.
- Recently, we've seen various early attempts at making Transformers – in particular BERT (Devlin et al., 2018) – smaller by removing attentional heads (Michel et al., 2019), quantisation and pruning (Cheong and Daniel, 2019; Sucik, 2019). In terms of actually down-sizing and accelerating the models, knowledge transfer using teacher-student knowledge distillation has led to the most attractive results (Mukherjee and Awadallah, 2019; Tang et al., 2019b; Jiao et al., 2019; Sanh et al., 2019).
- However, these studies focus only on using knowledge distillation as a tool. Important questions about the nature of this technique and how it interacts with properties of the teacher and student models remain generally unexplored.
- In line with the increasing demand for explainable AI, it is desirable that, for the beginning, at least the researchers better understand the tools they use, in this case distillation of NLP knowledge from Transformer models. Indeed, such understanding is also useful for overcoming the limitations and designing new variants of this method for smaller and better classifiers.

1.2 Aims

I aim to better understand knowledge distillation by exploring its use for knowledge transfer from BERT into different student architectures on various NLP tasks.

This can be further broken down into three aims:

- Explore the effectiveness of knowledge distillation in very different NLP tasks. To cover a broad variety of tasks, I use sentence classification datasets ranging from binary sentiment classification to 57-way intent classification to linguistic acceptability.
- Explore how distilling knowledge from a Transformer varies with different student architectures. I limit myself to using the extremely popular BERT model (Devlin et al., 2018) as the teacher architecture. As students, I use two different architectures: a BiLSTM, building on the successful work of Ralph Tang (Tang et al., 2019b,a), and a down-scaled BERT architecture.
- Explore how successfully can different types of NLP knowledge and capabilities be distilled. Since NLP tasks are often possible for humans to reason about, I analyse the models' behaviour (e.g. the mistakes they make) to learn more about knowledge distillation. I also probe the models for different linguistic capabilities, inspired by previous successful probing studies (Conneau et al., 2018; Tenney et al., 2019a).

1.3 Contributions

My actual findings. To be added later.

Chapter 2

Background [REVIEWED]

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 some 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,

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

predicting its next word, and 2) given a word from a sentence, predicting the words preceding the given one in the same sentence². 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 Lample 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., 1986) 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. A common downside of RNNs is that they “forget” over longer sequences. This

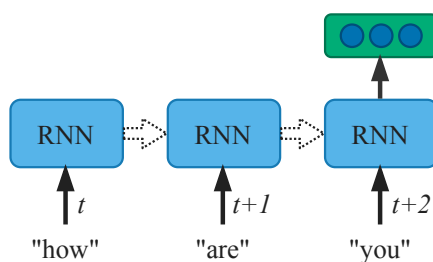


Figure 2.1. A recurrent neural network (RNN) consumes at each timestep one input word. Then, it produces a single vector representation of the inputs.

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. (2014) 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:

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

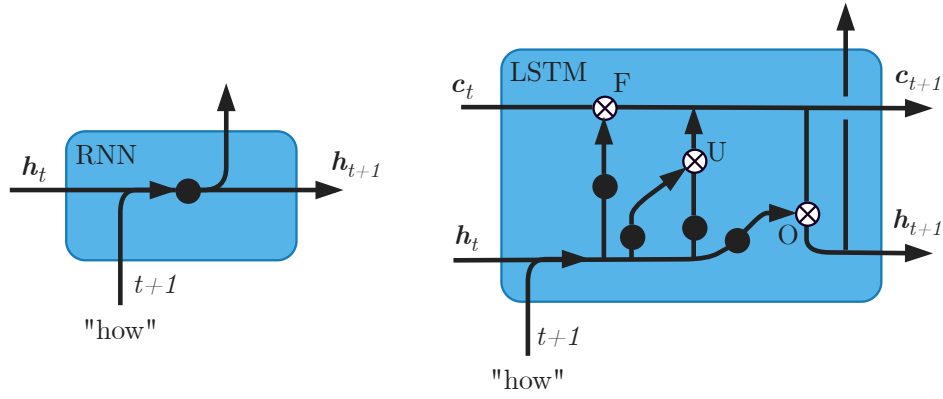


Figure 2.2. Comparing the internals of a vanilla RNN and an LSTM. The latter has three gates (shown as \otimes) – the forget gate F , the update gate U , and the output gate O . h is the internal (hidden) state which can be used as the output at any timestep, c is the memory cell. With \bullet is shown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

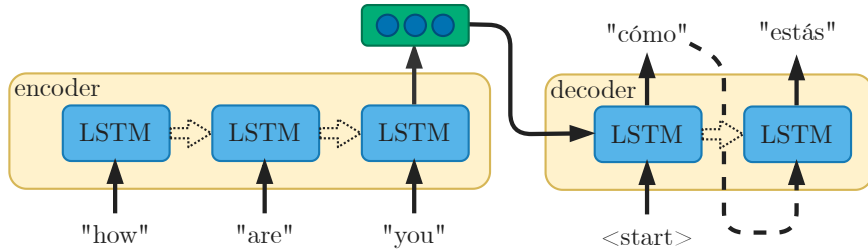


Figure 2.3. An encoder-decoder model for machine translation. Notice how the decoder initially takes as input the special $\langle \text{start} \rangle$ token and at later time consumes the previous output word.

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 formally³: First, the decoder state \mathbf{h}_D is processed into a *query* \mathbf{q} using a learnable weight matrix W_Q :

$$\mathbf{q} = \mathbf{h}_D W_Q \quad (2.1)$$

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

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

³My description does not exactly follow the original works of Bahdanau et al. (2014) and Luong et al. (2015). Instead, I introduce concepts that will be useful in later sections of this work.

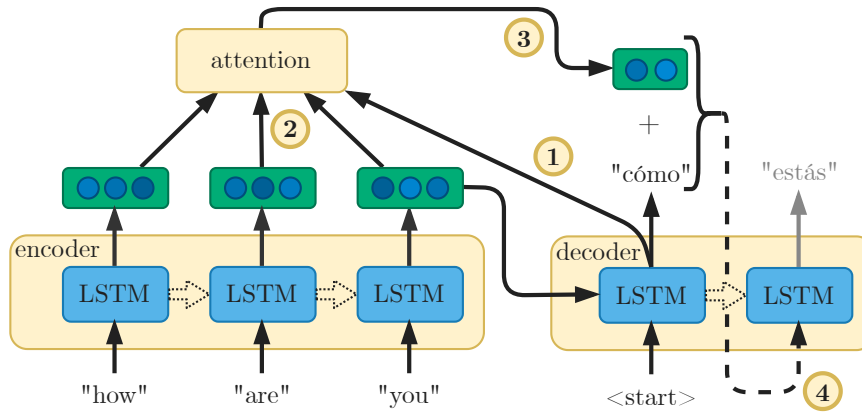


Figure 2.4. An encoder-decoder model for machine translation with added attention mechanism.

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:

$$w^{(i)} = \mathbf{q}^\top \mathbf{k}^{(i)} . \quad (2.3)$$

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

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

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).

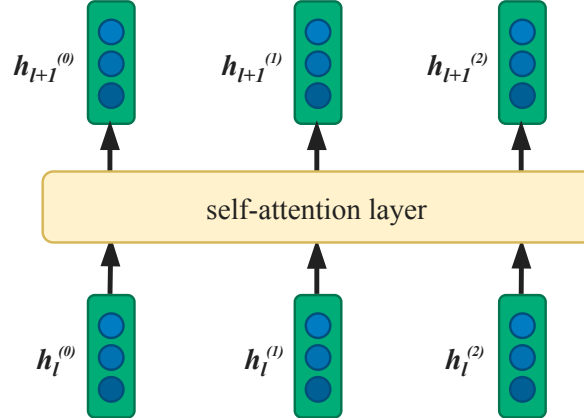


Figure 2.5. A high-level diagram of the application of self-attention in Transformer models. Three hidden states are shown for consistency with the length of the input shown in Fig. 2.4; in general, the input length can vary.

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 $\mathbf{h}_{l+1}^{(i)}$ in layer $l+1$ of the i -th input word, self-attention uses the incoming representation $\mathbf{h}_l^{(i)}$ for the query, and considers focusing on all representations in layer l , including $\mathbf{h}_l^{(i)}$ itself. Fig. 2.6 shows this in detail for input position $i=0$. Query $\mathbf{q}^{(0)}$ is produced and matched with every key in layer l (i.e. $\mathbf{k}^{(0)}, \dots, \mathbf{k}^{(2)}$) to produce the attention weights. These weights quantify how relevant each representation $\mathbf{h}_l^{(i)}$ is with respect to position $i=0$. Then, the new contextual embedding $\mathbf{h}_{l+1}^{(i)}$ is constructed as a weighted sum of the values $\mathbf{v}^{(0)}, \dots, \mathbf{v}^{(2)}$ (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 $\mathbf{h}_{l+1}^{(0)}, \dots, \mathbf{h}_{l+1}^{(2)}$ 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.

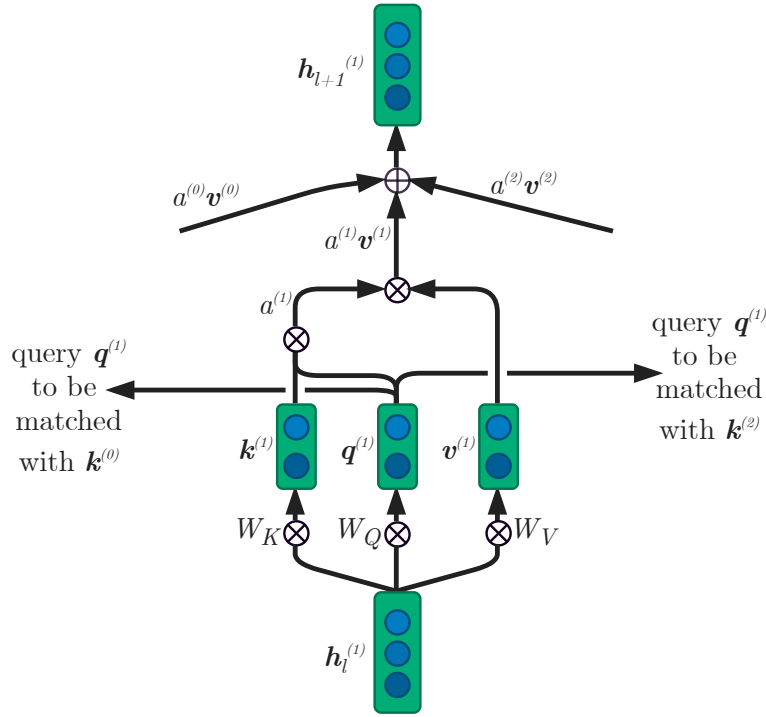


Figure 2.6. The internals of self-attention, illustrated on creating the next-layer hidden representation of the input position $i = 1$, given all representations in the current layer (previous, current, and following). Note that \otimes stands for multiplication (where the multiplication involves a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

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

$$O_{att,a} = \text{softmax}\left(\frac{qk^\top}{\sqrt{d_k}}\right)v = \text{softmax}\left(\frac{(h_l^\top W_{Q,a})^\top (h_l^\top W_{K,a})}{\sqrt{d_k}}\right)(h_l^\top W_{V,a}) \quad (2.5)$$

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 h_{l+1} in Fig. 2.6:

$$O_{att} = [O_{att,1}, \dots, O_{att,A}]W_{AO} . \quad (2.6)$$

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)⁴, the general procedure is:

⁴The 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. \(2018\)](#). 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 out⁵, 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.

⁵I.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. \(2018\)](#)). 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 pieces⁶. 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 representation, 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. \mathbf{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 \mathbf{o}_3 to predict the correct token at this position).

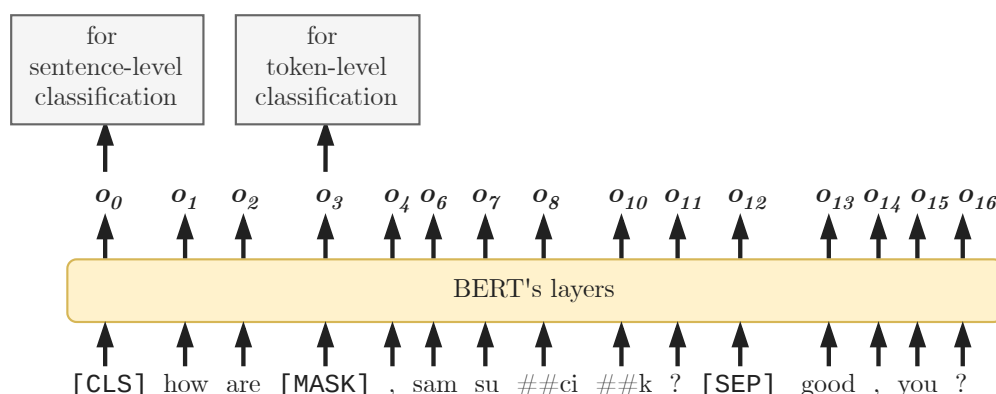


Figure 2.7. BERT’s handling of input for sentence-level and token-level tasks. The input sentences ($S_A = \text{How are you, Sam Sucik?}$ and $S_B = \text{Good, you?}$) are shown as split by BERT’s tokeniser, with the first instance of “you” masked out for MLM.

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 \mathbf{o}_0 as discussed above) and applies a fully-connected

⁶In 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.

layer with the tanh activation function. The external classifier is often another fully-connected layer with the tanh activation, producing the logits⁷. These get normalised using softmax to produce a probability distribution over all classes. The most probable class get output as the model’s prediction.

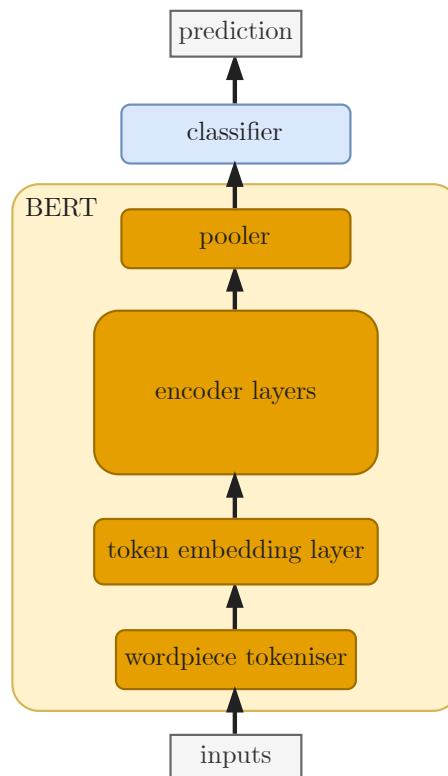


Figure 2.8. High-level overview of the modules that make up the architecture of BERT as used for sequence-level classification.

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

⁷For a classifier, the logits are the (unnormalised) predicted class probabilities.

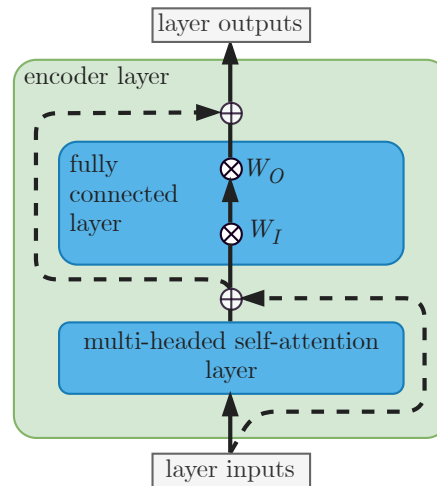


Figure 2.9. The modules making up one encoder layer in BERT; residual connections highlighted by using dashed lines. \otimes marks learnable neural layers, \oplus marks summation (in this case used to combine residual information with layer outputs).

fully connected layer output (see Fig. 2.9 for the summation points), and to the output 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: BERT_{Base} with 110 million parameters, 12 encoder layers and 12-head self-attention, and BERT_{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., 2019). 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., 2018), 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).
- Lample and Conneau (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:

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

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.

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

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

The temperature parameter T determines the extent to which the distribution will be “unsharpened” – two extremes being the completely flat, uniform distribution (for $T \rightarrow \infty$) and the maximally sharp distribution⁸ (for $T \rightarrow 0$). 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: $\frac{\partial \mathbf{o}_s}{\partial \mathbf{x}} - \frac{\partial \mathbf{o}_t}{\partial \mathbf{x}}$ (\mathbf{x} being an input, e.g. a sentence, and \mathbf{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”

⁸I.e. having the preferred class’s probability 1 and the other classes’ probabilities 0.

distillation: First, knowledge is distilled from the teacher into an intermediate-size “teacher assistant” model, then from the assistant into the final student.

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 revolutionary 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 distill. In particular, one can distill 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 distill 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 is successfully fine-tuned for a wide range of tasks, it is only 40% smaller than the BERT_{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) really understanding the models 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 program⁹, 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 workshops¹⁰.)

The area of image processing has seen the most attempts at interpreting neural models and their behaviour. One reason being that vision 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., 2013), or highlighting those parts of an image that a particular neuron “focuses” on (Zeiler and Fergus, 2013).

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 (2018) observe that many methods for analysing and interpreting models are simply adapted from image processing, in particular 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. (2018)). However, these methods are mostly qualitative and suitable for exploring individual input examples, thus not well suited for drawing statistically backed conclusions or for 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

⁹www.darpa.mil/program/explainable-artificial-intelligence

¹⁰See sites.google.com/view/xai2019/home, xai.kdd2019.a.intuit.com/, explainai.net/.

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 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 Conclusions [READY FOR REVIEW]

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 [REVIEWED]

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 tasks¹ is the GLUE benchmarking collection (Wang et al., 2018). This collection comprises 11 tasks which enable model benchmarking on a wide range of NLP tasks 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 leaderboard².

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 Accept-

¹Challenging by the nature of the tasks and by the small dataset size.

²gluebenchmark.com/leaderboard

ability (CoLA) and the Stanford Sentiment Treebank in its binary classification variant (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 Rasa³, 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., 2018](#)) 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.* (unacceptable⁴)
- *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.

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).

³[rasa.com](#)

⁴The “*” is a standard way to mark ungrammatical sentences in linguistic literature.

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 segments⁵, 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%⁶.

3.1.3 Sara [READY FOR REVIEW]

As the third task, I use an intent classification dataset created by Rasa, a start-up building open-source tools for conversational AI⁷.

The dataset is named Sara after the chatbot deployed on the company’s website⁸ 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). 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 for each 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](#).

⁵This is due to the use of an automated parser in creating the dataset.

⁶See the GLUE leaderboard at gluebenchmark.com/leaderboard

⁷For 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.

⁸See the bot in action at rasa.com/docs/getting-started/.

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 from conversations with the website’s visitors⁹. 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 (accessed April 3, 2020). It contains both the original hand-crafted examples as well as the (much more abundant) “real” examples.

The Sara dataset version I use dates back to October 2019, when I obtained it from Rasa and pseudonymised the data¹⁰. 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 ($F1_{micro}$), 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 : $F1 = \frac{2PR}{P+R}$. 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:

$$F1_{macro} = \frac{1}{C} \sum_c F1_c = \frac{1}{C} \sum_c \frac{2P_c R_c}{P_c + R_c}, \quad C \text{ being the number of classes} \quad (3.1)$$

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 and hence 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 suggest, 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 $F1 = \frac{2PR}{P+R}$) and 0 for an incorrect prediction (by

⁹To get a 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/

¹⁰As a former employee of Rasa, I got access to the data under the NDA I had signed with the company. I had a permission from Rasa to use the pseudonymised data for this project; the use complied with the ethical approval process of Rasa.

definition):

$$F1_{micro} = \frac{1}{N} \sum_n F1_n = \frac{1}{N} \sum_n \begin{cases} \text{if correct} & 1 \\ \text{else} & 0 \end{cases}, \quad N \text{ being the number of examples} \quad (3.2)$$

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 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 [Section 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 set¹¹. 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.

¹¹I used the code for [Tang et al. \(2019a\)](#) which is available at github.com/castorini/d-bert (accessed April 4, 2020).

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.

3.3 Probing tasks

The probing tasks (discussed in Section 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 tasks¹².

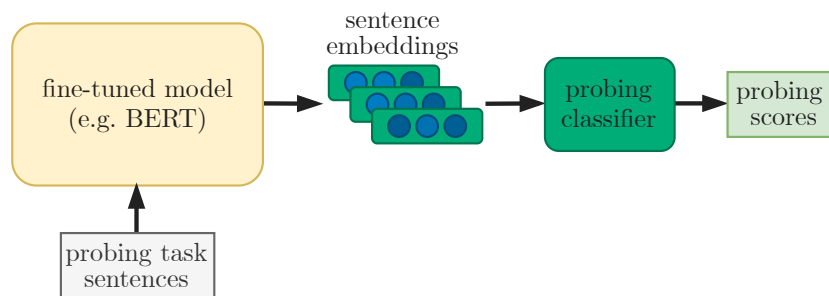


Figure 3.1. A high-level diagram of the probing process.

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 hyperparameters 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:

¹²The 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 (accessed April 4, 2020).

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 chosen such that each contains exactly one of these words. The task is 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 binary classification task, identifying the tense (present or past) of the sentence – given by the main verb of the sentence (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 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 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 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 Conclusions [READY FOR REVIEW]

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 focus 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

[REVIEWED]

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 bi-directional 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 previously reported scores or on finding the best hyperparameter configurations; I aim to learn more about knowledge distillation.

Being inspired by my internship at Rasa on compressing BERT¹, 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 [Section 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.

¹See blog.rasa.com/compressing-bert-for-faster-prediction-2/ and blog.rasa.com/pruning-bert-to-accelerate-inference/, accessed April 4, 2020.

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.

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.

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,

²github.com/huggingface/transformers, accessed April 4, 2020

³github.com/huggingface/transformers/tree/master/examples/distillation, accessed April 4, 2020

⁴github.com/castorini/d-bert, accessed April 4, 2020

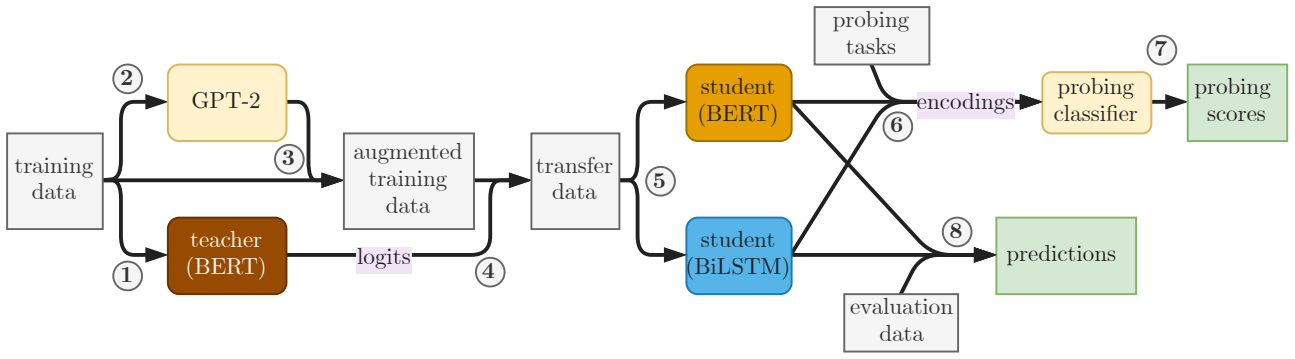


Figure 4.1. The main pipeline of this work: ① teacher fine-tuning, ② GPT-2 fine-tuning, ③ generating augmentation sentences, ④ adding teacher logits to the augmented training dataset, ⑤ knowledge distillation into students, ⑥ producing probing sentence encodings, ⑦ training the probing classifier and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

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 Section 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. (2018)) and is therefore more attractive, but also slower, with a greater incentive for compression.

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 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

⁵github.com/facebookresearch/SentEval

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
batch size	36
dropout rate	0.1

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

converges on Sara: 10. See Fig. 4.2 for the evaluation-set performance of the teacher models and how they converge during fine-tuning.

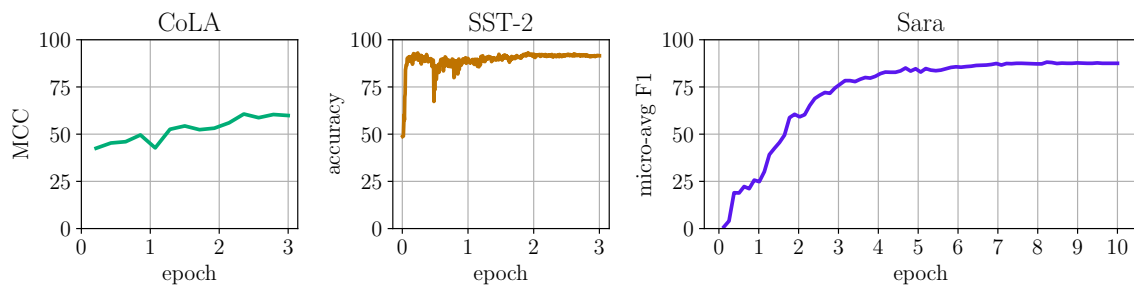


Figure 4.2. The evaluation-set performance of teacher models across fine-tuning. Unintentionally, I used different logging frequencies in fine-tuning the teachers, hence the SST-2 plot is dense (and appears more noisy) while the CoLA plot is sparse.

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 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.

4.3.3 BiLSTM student model

As the first student, I use the bi-directional 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

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.

last hidden states for either of the two processing directions are concatenated and passed to a fully connected layer with 400 output units⁶, 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.

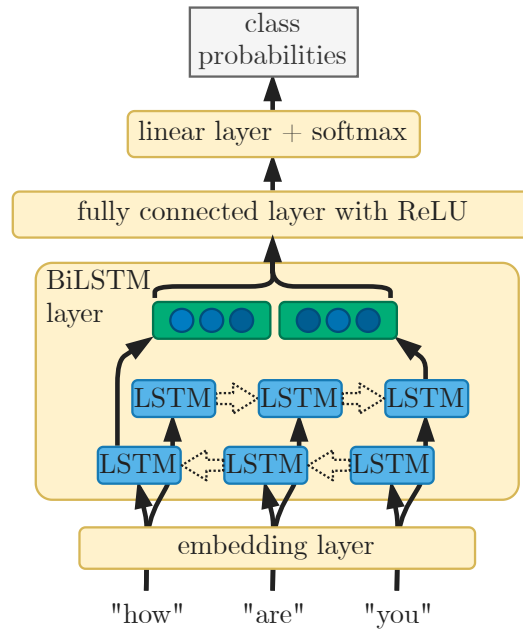


Figure 4.3. The bi-directional LSTM student. Diagram adapted from Figure 1 of Tang et al. (2019b).

The original model was built to process sentences word by word, encoding each word using the pre-trained word2vec embeddings⁷ 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.

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

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

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 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), 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 $T = 3$.⁸

⁸Usual values are from 1 (no effect) to 3. For instance, Sanh et al. (2019) use $T = 2$. In a work that is much close 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_S, which was initialised from word2vec. Later, I explore different ways of initialising the embedding layers.

During training, LSTM_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_S (same as Tang et al.). BERT_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 Adadelata learning algorithm (Zeiler, 2012) is used for training LSTM_S, with $\eta = 1.0$ and $\rho = 0.95$, while Adam is used for BERT_S. Note that Adam is an improved successor of Adadelata⁹ and is much more widely used; later in this work, I explore the use of Adam for LSTM_S. No η scheduling is used with LSTM_S, while BERT_T uses scheduling similar to BERT_T. To prevent “gradient explosion” in LSTM_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. (2018) and Tang et al.. Tang et al. report the small batch size $B = 50$ to work well with the BiLSTM student. For BERT_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_T allows sequences of up to 512 word-pieces 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_S.

	LSTM _S	BERT _S
loss function	mean square error	cross-entropy, $T = 3$
learning algorithm	Adadelata ($\eta = 1.0$, $\rho = 0.95$)	Adam ($\eta = 5 \times 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.98$)
training budget	30 epochs	60 epochs
η scheduling		linear warm-up (10 epochs), then linear decay
batch size	50	256
embedding layer initialisation	word2vec	wordpiece, from BERT _T

Table 4.3. The initial parameters of both student models.

⁹In particular, the adaptive mechanism of Adadelata 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]).

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 cluster¹⁰.

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](#)¹¹. The reason why all 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

¹⁰computing.help.inf.ed.ac.uk/teaching-cluster.

¹¹Note 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.

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.

4.5 Conclusions [READY FOR REVIEW]

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

[READY FOR REVIEW]

In this chapter, I use knowledge distillation to train student models BERT_S and LSTM_S from the fine-tuned teacher BERT_T on each of the three downstream tasks. My aim is to end up with students that are small but perform well. Ideally, the student size will stay at the initial 2.4 millions of trainable non-embedding parameters while the evaluation-set performance will be above 90% of the teacher’s performance.

As discussed in [Section 4.1](#), my aim is not to find the best possible student hyperparameters, but I still 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 configuration of each student on CoLA, looking for good evaluation-set score and fast convergence. 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, I adjust the size of each student separately for each task because some tasks are known to be more or less difficult than others, requiring the students to be more or less complex.

After obtaining well-performing students for each task, I compare them with one another and with the respective teacher and discuss my observations.

5.1 General hyperparameter exploration on CoLA

I restrict myself to exploring the following essential hyperparameters (in both students):

1. η – the learning rate. Additionally, for LSTM_S , I compare the original Adadelta learning algorithm with the more general Adam.
2. Learning rate scheduling, more concretely the warmup duration (in epochs) E_w of gradual warmup of the learning rate ([Goyal et al., 2017](#)), and the optional use of linear decay following the warmup.
3. B – the minibatch size.

4. Embedding type – word-level vs wordpiece.

Following on the discussion of implementation details in [Section 4.3](#), the initial configurations of the parameters to be explored are:

- For BERT_S: Adam with $\eta = 5 \times 10^{-5}$, $B = 256$, $E_w = 10$ – warmup over the first 10 epochs, followed by linear decay of η , and wordpiece embedding layer with the non-static mode.
- For LSTM_S: Adadelta with $\eta = 1.0$, $B = 50$, no η warmup or decay, and the word-level embeddings with the multichannel mode.

Note that the embedding mode is explored separately for each task in the next section.

Initially, BERT_S was initialised entirely from scratch, including the embedding layer. This may pose a disadvantage for BERT_S because the BiLSTM student starts with embeddings initialised from the pre-trained word2vec. To eliminate this disparity, I initialise BERT_S's wordpiece embeddings using the parameters from the fine-tuned BERT_T. Because the teacher's embeddings are high-dimensional (1024-D), a trainable linear layer is used inside BERT_S to project them to $d_h = 204$ dimensions¹. 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 student in knowledge distillation from Transformers in this way. In exploring the parameters of BERT_S, I report results for the variant with the teacher's embedding knowledge. Results for the variant initialised entirely from scratch are in **TODO: APPENDIX X**. In general, the difference is not large, but the version initialised from scratch performs slightly worse.

5.1.1 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. 5.1](#) 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 $\eta = 5 \times 10^{-3}$ “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.

¹The token type and positional embeddings are not initialised from the teacher and hence do not require dimensionality reduction. They are added to the wordpiece embeddings after these are dimensionality-reduced.

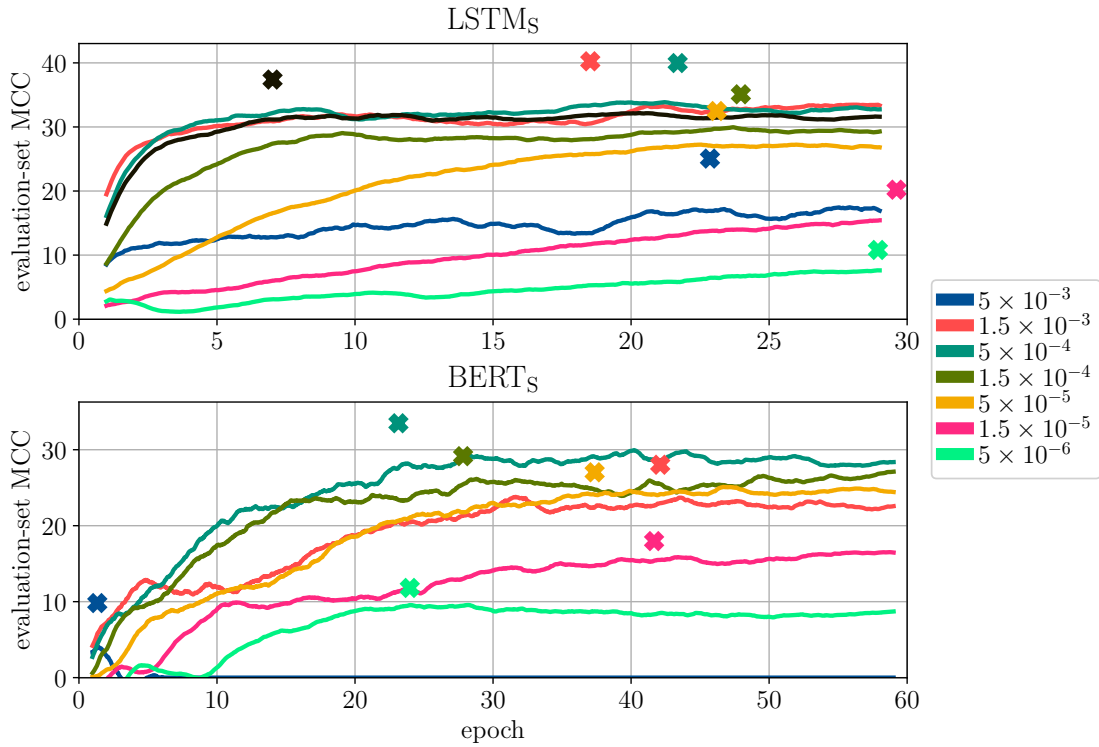


Figure 5.1. Comparing various η values on CoLA. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

In the case of LSTM_S, the Adadelata algorithm is outperformed by Adam. From now onwards, I use Adam with both students, in all cases with $\eta = 5 \times 10^{-4}$.

5.1.2 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.

As Fig. 5.2 shows, LSTM_S clearly does prefer small batch sizes. However, training with tiny minibatches takes very long – compare ~ 25 min for $B = 512$ with ~ 7 h for $B = 32$. Hence, I restrain from trying even smaller batch sizes and use $B = 32$ for LSTM_S in all further experiments.

Fig. 5.3 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.

In the case of LSTM_S, Fig. 5.3 shows that the longer the warmup duration, the slower the model converges. This is understandable because during warmup, learning happens less

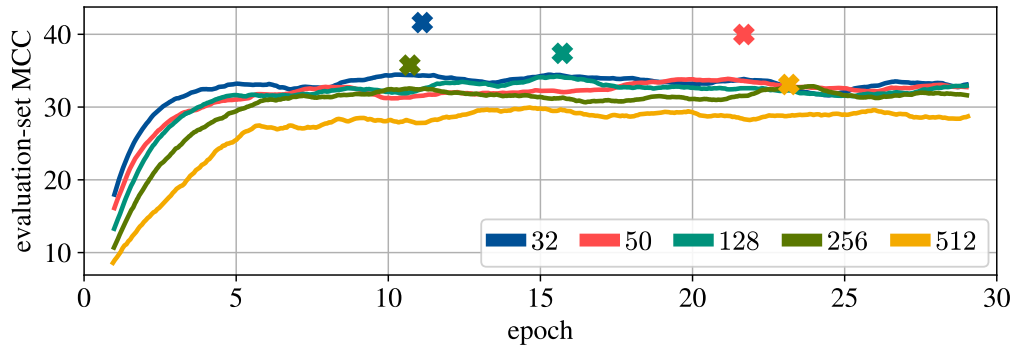


Figure 5.2. Comparing various batch sizes for LSTM_S on CoLA. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

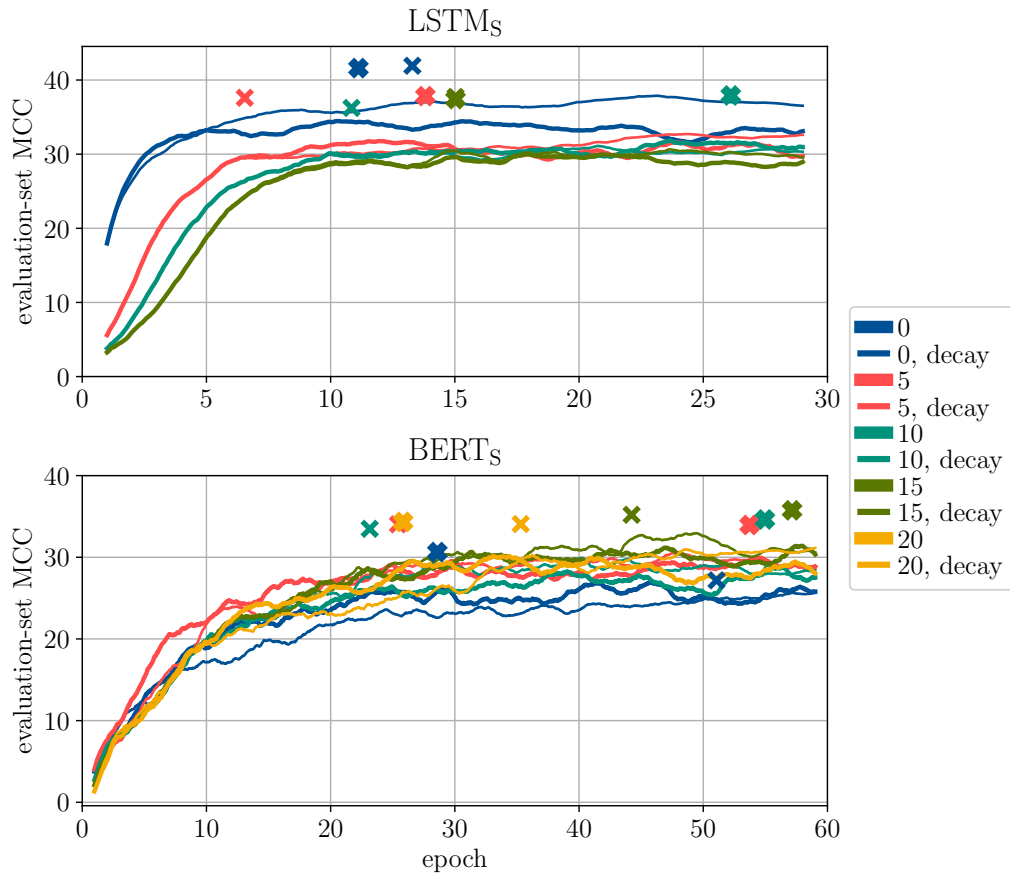


Figure 5.3. Comparing warmup durations E_w on CoLA, with the optional learning rate decay to 0 over the remaining training epochs. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

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. 5.3 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. 5.3 in later epochs (beyond epoch 35).

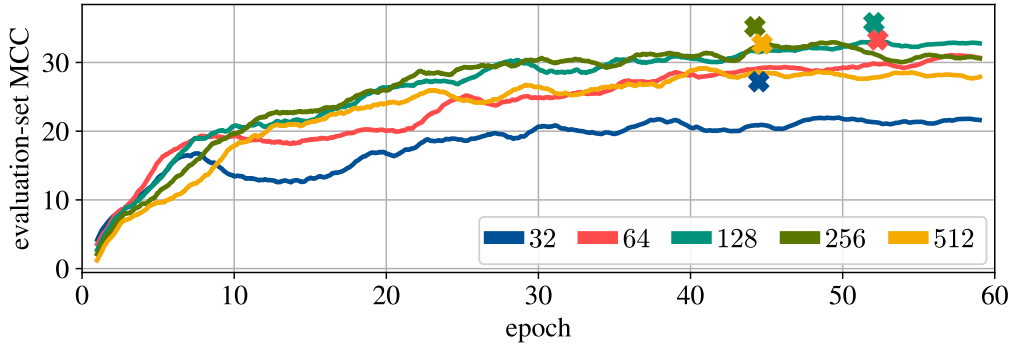


Figure 5.4. Comparing various batch sizes for BERT_S on CoLA. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

The batch size exploration in Fig. 5.4 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.

5.2 Optimising students for each downstream task

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

5.2.1 Choosing embedding type and mode

In the previous section, LSTM_S consistently outperformed BERT_S. This can be due to different model architectures or due to different embedding types and modes – while LSTM_S has its word-level embedding layer initialised using word2vec and uses the multi-channel mode², BERT_S uses wordpiece-level embeddings from the fine-tuned BERT_T in the non-static mode³.

²I.e. uses two parallel embedding table instances, one frozen and the other one allowed to further change in training.

³This is the usual case where a single embedding table instance is used and allowed to change during training.

I explore combinations of the two embedding types and of different embedding modes. Following preliminary experiments, I do not use the static (embedding freezing) mode as it has detrimental effects on student learning. Left for exploration are the two modes that do allow embedding parameters to be tuned during training: the multichannel and non-static modes.

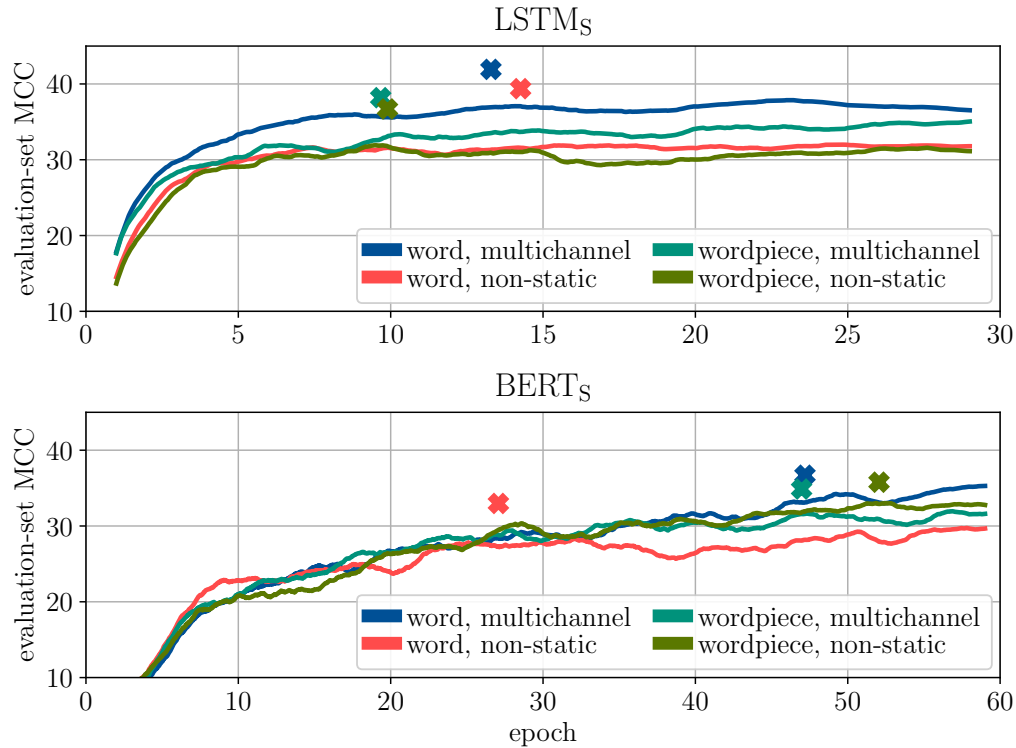


Figure 5.5. Comparing embedding types and modes on CoLA. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

Fig. 5.5 shows how different type and mode combinations affect knowledge distillation on CoLA. With LSTMs, 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 BERTs, 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. 5.5. The results are shown in Fig. 5.6. 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. 5.6 are much smaller than the differences observed on CoLA in Fig. 5.5. 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. 5.6).

Results on Sara are shown in Fig. 5.7. Here again, the relative differences in performance

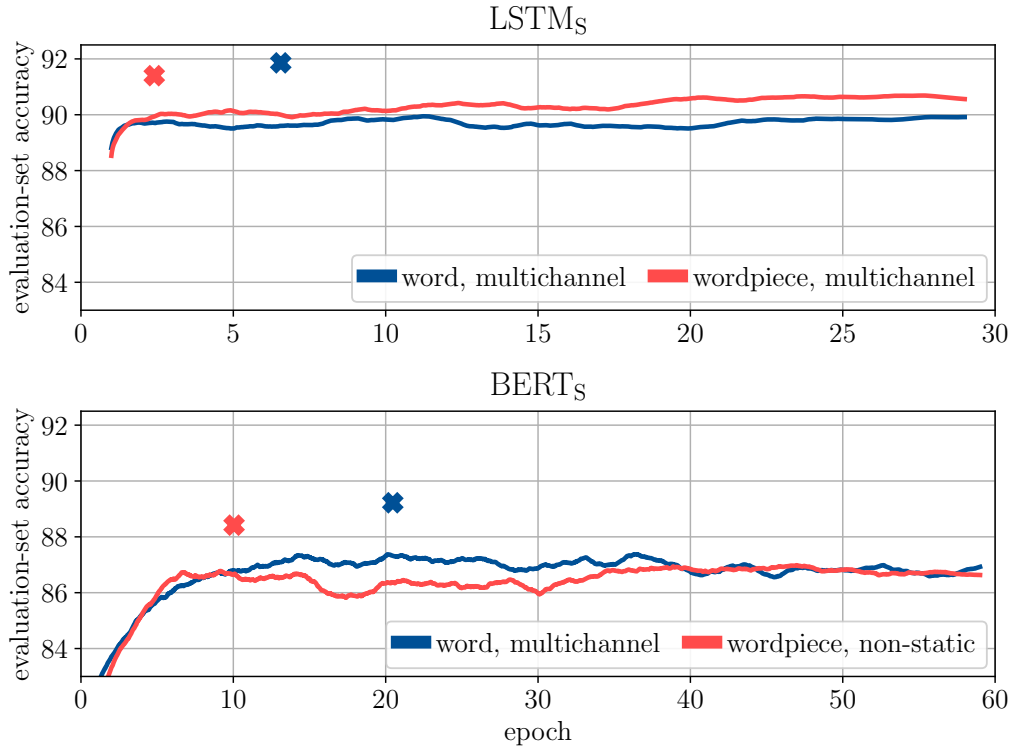


Figure 5.6. Comparing embedding types and modes on SST-2. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

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 .

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 word-piece embeddings have to be assembled from multiple pieces. On these words – frequent

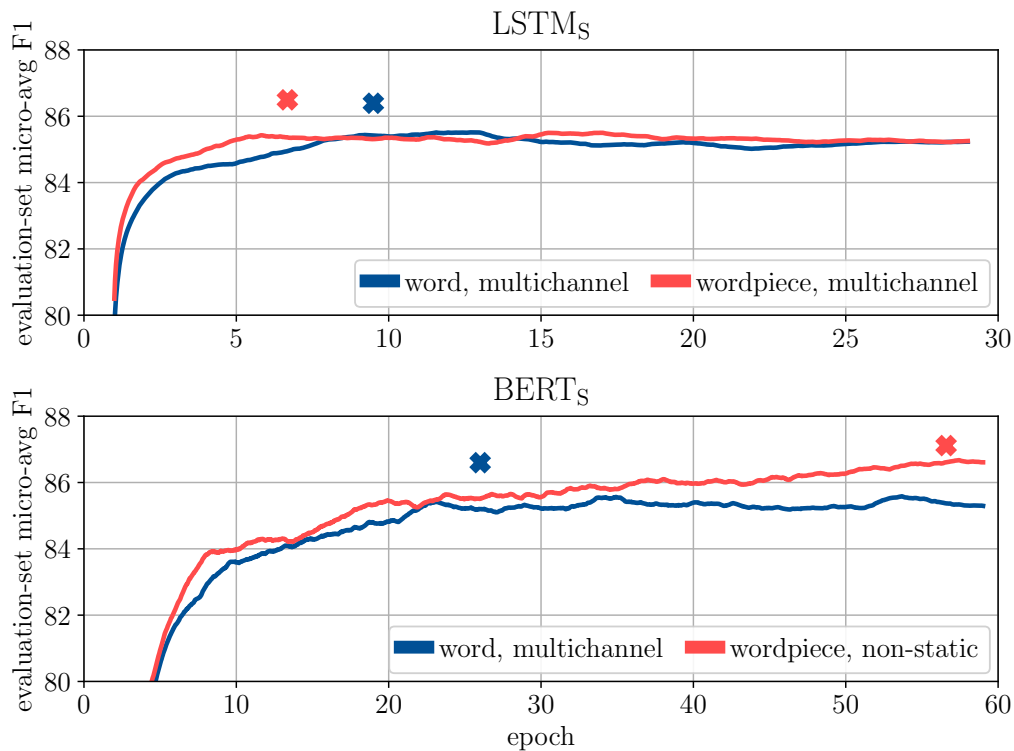


Figure 5.7. Comparing embedding types and modes on Sara. Crosses mark the maximum scores. The lines are smoothed using sliding average with a 2-epochs-wide window.

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.

5.2.2 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 LSTMs, 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. 5.1 (BERT_S) and in Tab. 5.2 (LSTM_S).

W	d_h	d_I	A		D	L
1/16	13	47	1		1/4	1
1/8	26	94	1		1/3	2
1/4	51	188	1		1/2	3
1/3	68	250	1		1	5
1/2	102	375	2		2	10
1	204	750	3		3	15
2	408	1500	6			
3	612	2250	9			
4	816	3000	12			

Table 5.1. BERT_S dimensions for different width scalings (left) and depth scalings (right). The default size with 2.4 million parameters corresponds to $W = 1$, $D = 1$.

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

Table 5.2. LSTM_S dimensions for different width scalings (left) and depth scalings (right). The default size with 2.4 million parameters corresponds to $W = 1$, $D = 1$.

5.2.2.1 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 example, see Fig. 5.8 where the gradient explosion happens around epoch 7 and the model score (MCC) then falls to 0 and stays there.

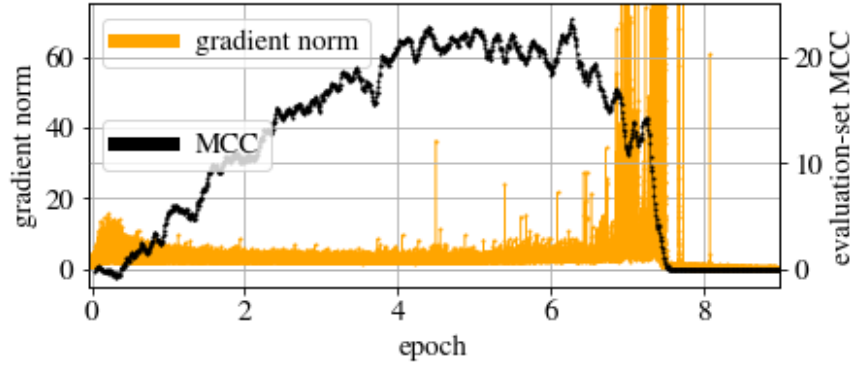


Figure 5.8. Gradient explosion in BERT_S with $W = 2$ and $D = 2$. The MCC values have been smoothed with a 0.1-epoch-wide sliding average window.

Without further extensive exploration of optimal learning rate values for each BERT_S size⁴, 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. 5.9. 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. 5.9, 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.

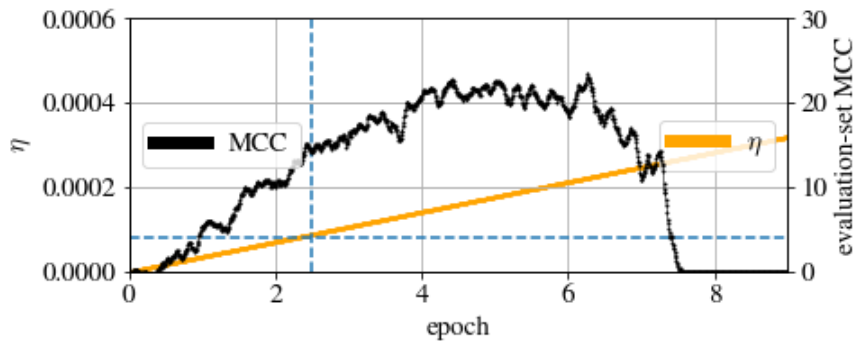


Figure 5.9. Learning progress (MCC over time) vs η for BERT_S with $W = 2$ and $D = 2$ – the model experiencing gradient explosion in Fig. 5.8. The dashed lines show the position I identify as the approximate latest point of training before the learning starts to slow down, and the learning rate at that position. The MCC values have been smoothed with a 0.1-epoch-wide sliding average window.

⁴This would be extremely time-consuming because the larger versions take over 3 days to train.

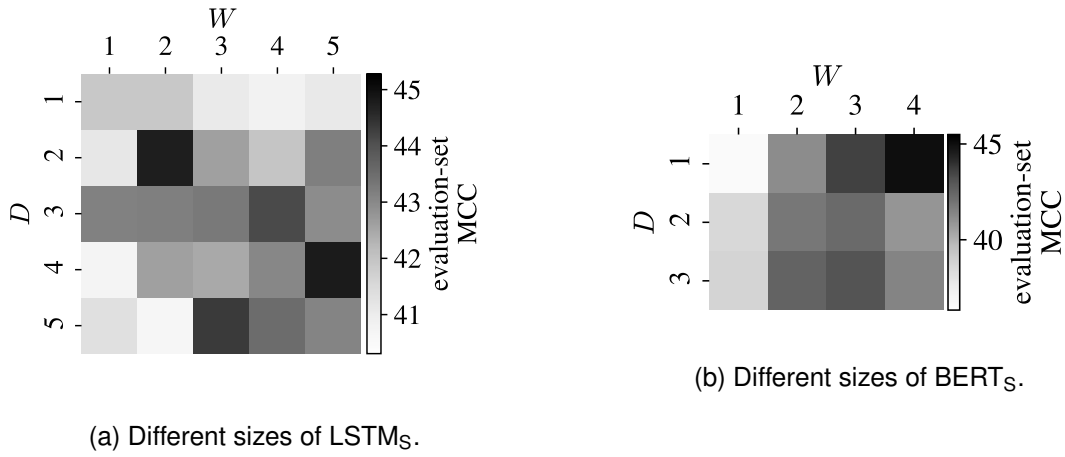


Figure 5.10. Best evaluation-set performance for the different student sizes on CoLA.

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. 5.10 presents the 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 approaching 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 $W = 2$, $D = 2$.

5.2.2.2 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. 5.11 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

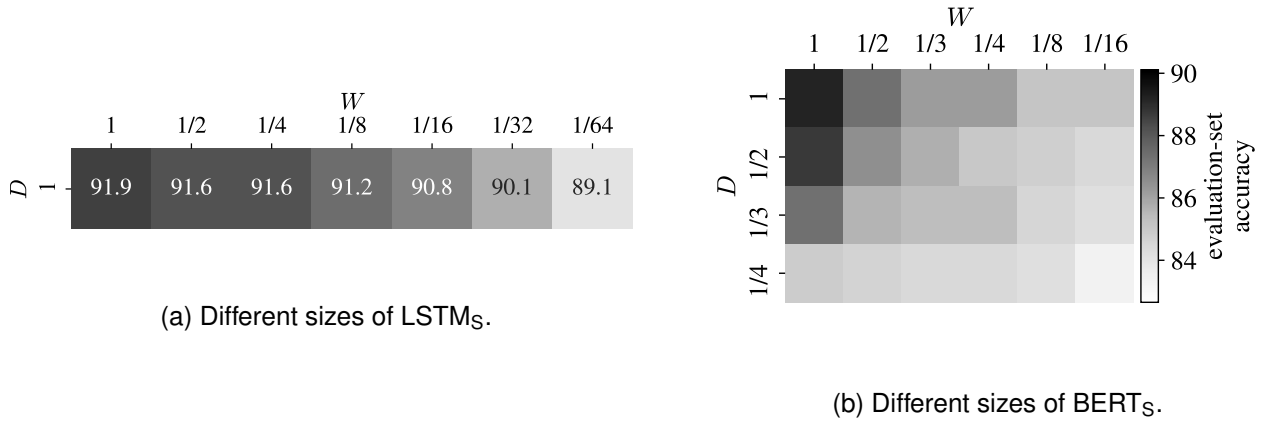


Figure 5.11. Best evaluation-set performance for the different student sizes on SST-2.

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. 5.11b 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.

5.2.2.3 Sara

Similar to SST-2, Sara is an easy task. With BERT_T achieving $F_{1micro} = 87.5$, 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. 5.12. 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 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.

5.3 Student training: conclusions

By conducting a brief exploration of the students' hyper-parameters, I produced well-performing BERT_S and LSTM_S for each of the three downstream tasks, and gained insights into the nature of the parameters, tasks and models.

CoLA is much more difficult of a task than SST-2 and Sara. Even with students having over 100 million parameters, they do not achieve more than 75% of the teacher's score.

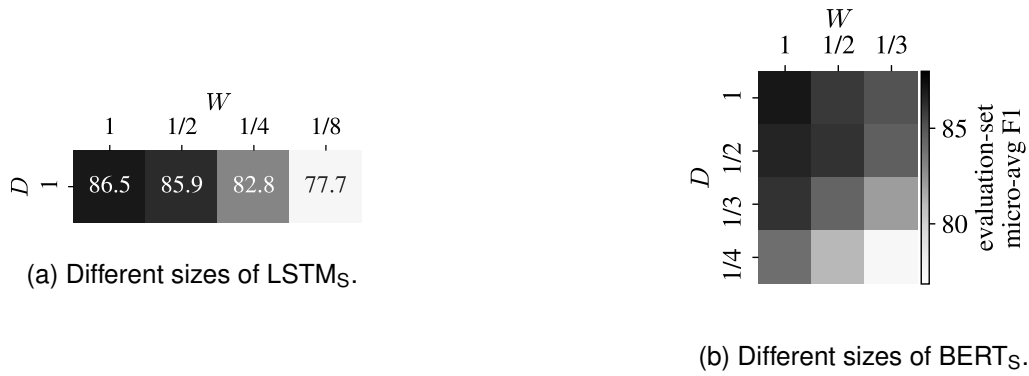


Figure 5.12. Best evaluation-set performance for the different student sizes on Sara.

Moreover, in general, BERT_S requires more parameters than LSTM_S for the same performance level. This can be attributed to the smaller initial width of BERT’s internal representations: 204 (compare with the 300- and 400-unit layers of LSTM_S).

There is further evidence of model width being more important than depth: In particular, the 5-layer BERT_S can be made shallower on both SST-2 and Sara without significant performance drops. I conclude that Transformer models like the 12- and 24-layer BERTs are unnecessarily deep for tasks like intent classification or even grammatical acceptability, and making them shallower is a promising way to accelerate the models. After all, this idea is in line with [Sanh et al. \(2019\)](#) who created well-performing BERT students shallower but not thinner than the teacher.

There are several differences between BERT_S and LSTM_S. Notably, the LSTM student converges much faster. Additionally, the much deeper BERT student is more sensitive to the learning rate values and these need to be significantly reduced for larger BERT_S sizes. While LSTM_S trains much faster, it works best with smaller minibatches – if larger batch sizes worked well, the training could be sped up even further.

The usual direction of knowledge flow into a student during knowledge distillation is top-down: Matching the teacher’s logits happens at the top of the students, and gradients trickle down into the lower student layers. Opposite and complementary to this is the provision of trained embeddings to the early layer of a student (knowledge injected at the bottom can trickle up during further training). I showed that both word-level word2vec embeddings and fine-tuned wordpiece embeddings taken from BERT_T work well with both student architectures. Importantly, certain downstream tasks like CoLA and SST-2 benefit from higher-quality word-level representations while others like Sara can benefit from 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 which reflects the initial knowledge injected before student training⁵

[Tab. 5.3](#) summarises the parameters of best students on each task. These 6 students, together with their teachers, I will use for further analysis. [Fig. 5.13](#) compares the best

⁵This corresponds to the multichannel embedding mode.

	model	size	width	L	η & scheduling	B	embeddings
CoLA	BERT _S	76.4M	$d_h = 816$, $d_I = 3000$, $A = 12$	5	$\eta = 7 \times 10^{-5}$, $E_w = 15$, decay	128	word2vec, multichannel
	LSTM _S	15.4M	$d_{LSTM} = 600$, $d_{FC} = 800$	2	$\eta = 5 \times 10^{-4}$, $E_w = 0$, decay	32	word2vec, multichannel
SST-2	BERT _S	2.4M	$d_h = 204$, $d_I = 750$, $A = 3$	5	$\eta = 5 \times 10^{-4}$, $E_w = 15$, decay	128	word2vec, multichannel
	LSTM _S	2.4M	$d_{LSTM} = 300$, $d_{FC} = 400$	1	$\eta = 5 \times 10^{-4}$, $E_w = 0$, decay	32	word2vec, multichannel
Sara	BERT _S	2.4M	$d_h = 204$, $d_I = 750$, $A = 3$	5	$\eta = 5 \times 10^{-4}$, $E_w = 15$, decay	128	wordpiece, non-static
	LSTM _S	2.4M	$d_{LSTM} = 300$, $d_{FC} = 400$	1	$\eta = 5 \times 10^{-4}$, $E_w = 0$, decay	32	wordpiece, multichannel

Table 5.3. Configuration for the best students on each task. The model size denotes the number of trainable, non-embedding parameters.

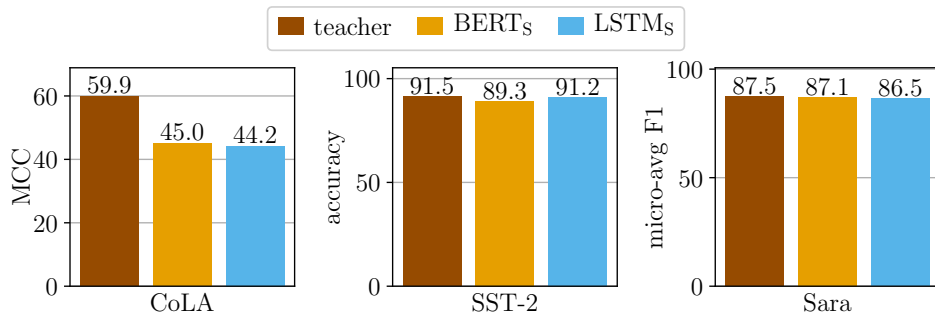


Figure 5.13. Evaluation-set scores of the teacher and the best students for each downstream task.

students with the teacher in terms of the evaluation-set score.

Chapter 6

Analysing the models

In this chapter, I use probing and prediction analysis to analyse, interpret and compare the teacher and the best student models for each downstream task. In doing so, I produce insights into the nature of the downstream tasks, the models, and knowledge distillation.

6.1 Probing [READY FOR REVIEW]

Tenney et al. (2019a) recently showed that a typical text processing pipeline can be identified in BERT. The early layers specialise in extracting simple properties like word classes (POS tags). Later layers extract more sophisticated information like dependencies (identifying the subject, object, and others). The last layers are capable of extracting complex semantic information such as real-world relations (predicting the relation between “Mary” and “work” in “Mary is walking to work.” to be of type `Entity-Destination`¹).

In the probing suite of Conneau et al. (2018), which I use, the pipeline steps are represented from the simplest ones to the most complex ones; from extracting surface properties like sentence length (`Length`) and concrete input words (`WordContent`), through extracting syntactic structure (`Depth`, `TopConstituents`), to identifying the subject, object and the main verb of a sentence and their properties (`Tense`, `SubjNumber`, `ObjNumber`), to detecting broken semantics (`OddManOut`, `CoordinationInversion`).

The probing suite itself is a tool which can be used in various ways. I use it primarily to trace language knowledge, to observe how the pre-trained knowledge of BERT is retained or changed in teacher fine-tuning, and subsequently distilled into student models. While Conneau et al. apply probing only to the final representations extracted from various sentence encoders, I adopt the approach of Tenney et al. and probe models at different layers. Thus, I not only quantify the linguistic knowledge, but also localise it within a given model².

¹Example borrowed from Tenney et al. (2019b, p. 4).

²As discussed in Section 4.3.6, in the case of LSTMs, I am only able to probe the final representations due to the way LSTMs are implemented in PyTorch.

By analysing the differences in linguistic knowledge between $BERT_T$, $BERT_S$, and $LSTM_S$ on a particular downstream task, I explore the effects of different student architectures as well as the kinds of knowledge that is easy or difficult for the students to learn. By observing differences between models trained on different downstream tasks, I explore the possibility of characterising a downstream task by the kinds of linguistic knowledge it does or does not require.

6.1.1 Probing teacher models

[Fig. 6.1](#) compares the probing results of the pre-trained BERT and of BERT further fine-tuned on different downstream tasks. First of all, the results are mostly very similar across all the models, which means that fine-tuning on downstream tasks did not change the models too drastically.

The models’s results deviate especially for the last layers while for early layers fine-tuning does not lead to any changes. In other words, only the last model layers get fine-tuned to the particular tasks, while the early layers effectively act as extractors of universally usable features. (It is therefore likely that the teacher models could be fine-tuned well even with the early layers frozen or only allowed to change during a small portion of the fine-tuning steps, thus making the fine-tuning faster.)

Another way of thinking about the teacher fine-tuning is that the pre-trained language knowledge is “replaced” by downstream task-specific skills. If the downstream task relies on the linguistic skills, then they are retained. This effect is clear in [Fig. 6.1](#) when comparing the CoLA teacher with the SST-2 and Sara ones. As a task about linguistic acceptability, CoLA requires models to be sensitive to a wide range of linguistic principles and their violations. Therefore, very little of the pre-trained language knowledge is lost in fine-tuning – on the **BigramShift** task, the CoLA $BERT_T$ even outperforms the pre-trained model. Sentiment classification, on the other hand, requires capabilities very different from typical linguistic skills, leading to the last layers of the SST-2 $BERT_T$ retaining little of the pre-trained skills. Intent classification leads to retaining some of the pre-trained knowledge; arguably, skills like distinguishing between greetings, statements, commands and questions, or understanding that two sentences are semantically equivalent, are desirable for a task like Sara.

Interestingly, on **WordContent**, the CoLA teacher “forgets” more than the Sara teacher. I explain this in terms of certain Sara intents strongly linked to specific words – for example, examples of the intent **affirm** mostly identified by their use of “yes” or “okay”. Therefore, a good model for Sara will likely “remember” the exact input words even in higher layers.

Last but not least, the curves for each probing task in [Fig. 6.1](#) show how different linguistic skills are localised in the usual way. The two surface property skill (**Length**, **WordContent**) are localised in the first layers which have access to the input. The syntactic skills – **Depth** and **TopConstituents** – are localised in the middle layers. Finally, the semantic tasks – **OddManOut** and **CoordinationInversion** are best handled by the last layers.

An interesting case is the **BigramShift** task which, as previously discussed, is both syntactic and semantic, because disturbing the word order in a sentence typically disrupts

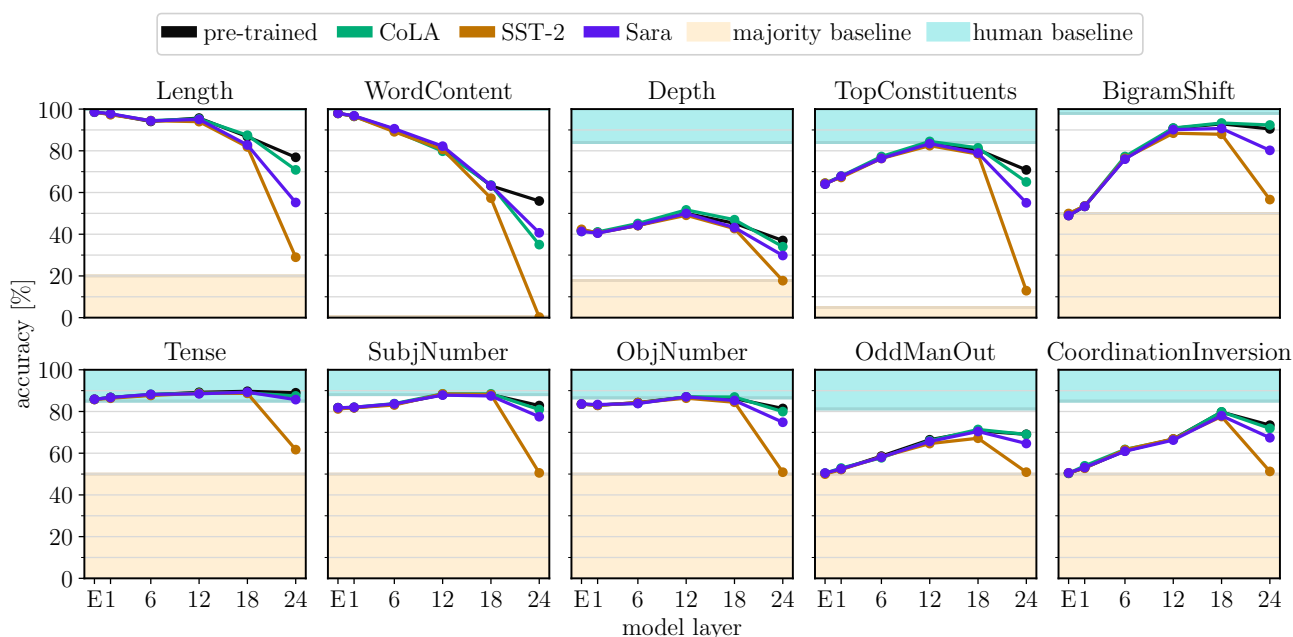


Figure 6.1. Probing results for the pre-trained BERT and the teacher BERT models. Probing was applied to encoder layers 1, 6, 12, 18, and 24, and to the embeddings (layer “E”) extracted just before the first encoder layer. The two baselines, bounding the expected model performance from below and from above, are the majority-class baseline and the human performance baseline as reported by [Conneau et al. \(2018\)](#).

both the meaning and the sentence structure structure. As a result, this probing task is handled well by the middle layers as well as the later ones.

As for *Tense*, *SubjNumber* and *ObjNumber*, they are handled well by all layers, but by the later ones in particular. I hypothesise that the overall good performance is due to the morphological component of the tasks. Statistically, in most cases, the tense of the main verb in English can be determined just by checking if a word in the sentence ends with “-d/-ed” (the regular-verb past tense marker) or not. Similarly, the number of the subject or object can be guessed just from knowing if any word ends in “-s” (the plural marker). Since morphological knowledge is known to be contained in pre-trained embeddings, even the embedding layer output performs well on these tasks. Minor improvements can be made by considering semantic and syntactic knowledge in cases where it is important to know which word of the sentence is the object, subject or main verb, or in cases where the markers are unusual (irregular verbs, unusual plural forms or singular forms ending in “-s”). This is in line with the later layers performing slightly better than the early ones.

6.1.2 Probing student models

[Fig. 6.2](#) shows the results of probing the best LSTM_S and BERT_S for each downstream task. To put the scores into perspective, I also include results achieved by probing the students’ embeddings *before* student training: This means pre-trained word2vec embeddings

for CoLA and SST-2³, and wordpiece embeddings from BERT_T for Sara.

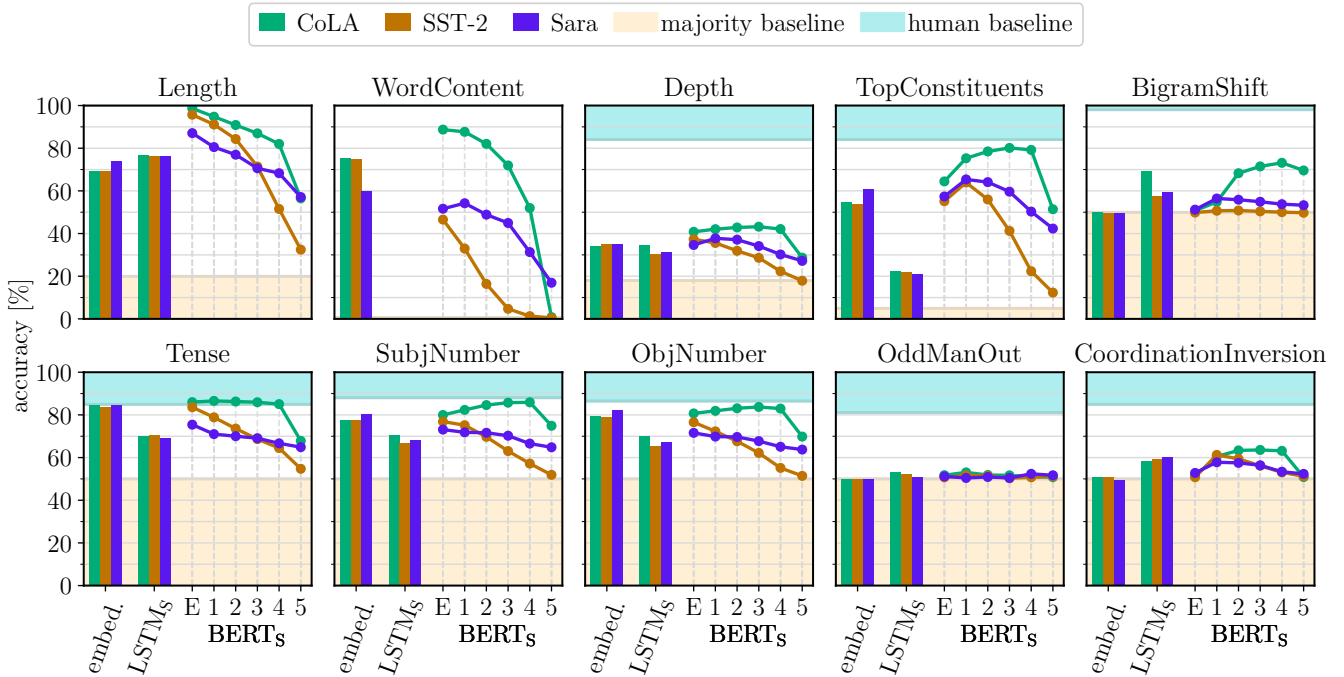


Figure 6.2. Probing results for the best student models. For comparison, results achieved just by using embeddings are shown as well (labelled “embed.”). While for LSTM_s, the probing encodings were extracted only after the last LSTM layer, for BERT_s, I extracted encodings after each of the 5 encoder layers and embedding encodings from before the first layer (labelled “E”). 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.

The first significant difference when comparing Fig. 6.2 with Fig. 6.1 is the progressive decrease in probing scores of BERT_s from earlier to later layers (in BERT_T, the scores only fall in the last layers). This is mostly visible in Fig. 6.2 for **Length**, **WordContent**, **Tense**, **SubjNumber** and **ObjNumber**. The decreasing scores reflect the different ways in which BERT_T and BERT_s are initialised. The teacher starts with all layers pre-trained (i.e. containing general language knowledge). During teacher fine-tuning, it is the last layer through which feedback enters the model. Therefore, the language knowledge is disturbed mostly in the last layers. The students, on the other hand, start with pre-trained knowledge “injected” only from the bottom – in the embedding layer. During student training (knowledge distillation), learning feedback enters from the top (in the last layer), but, at the same time, the embedding knowledge propagates up, affecting the earliest layers the most. Thus, the decreasing probing scores can be interpreted as markers of linguistic knowledge which was present in the initial embeddings, but the model did not retain it in the later layers. Either the knowledge was not needed for the downstream task, or the knowledge was only useful in creating more general representations in the first layers.

In general, both students perform worse than the respective teachers on all probing tasks.

³As previously discussed, only the embeddings corresponding to words found in the transfer dataset are taken.

This is not surprising as the students are smaller and lack the pre-trained knowledge that the teachers contain before fine-tuning. The students underperform especially on three tasks: **BigramShift**, **OddManOut**, and **CoordinationInversion**. These are difficult tasks, not solvable just using embeddings (in Fig. 6.2, the embeddings perform at the bottom baseline level). In other words, the students need to learn these linguistic skills by themselves, which they do with mixed success; in particular, all students fail to recognise semantically disturbed sentences in **OddManOut**.

Comparing the probing scores of one student architecture trained on different downstream tasks, there are several important observations. Firstly, different instances of **LSTM_S** perform similarly. This is likely due to the student being shallow, therefore the **LSTM** layer(s) producing relatively similar, task-agnostic representations, and only the fully-connected layer being heavily task-specific. Secondly, the **CoLA** task clearly encourages **BERT_T** to learn a wide range of linguistic knowledge (much more strongly than for **SST-2** and **Sara**): be it syntax (**Depth**, **TopConstituents**, **BigramShift**), morphology and dependencies (**Tense**, **SubjNumber**, **ObjNumber**), or semantics (**BigramShift**, **CoordinationInversion**). Thirdly, as previously observed for different teachers, **SST-2** requires the least linguistic knowledge and typically leads to the lowest probing scores.

When comparing the two student architectures trained on the same downstream tasks, it is difficult to observe any clear trends. **LSTM_S** performs comparably with the last **BERT_S** layers (**BigramShift**, **WordContent**, **OddmanOut**), on some other tasks **LSTM_S** roughly compares to the middle layers of **BERT_S** (**Length**, **Depth**, **Tense**), and still other times **LSTM_S** performs much worse than any layers of **BERT_S** (**TopConstituents** for **CoLA** and **Saras**). On **WordContent**, **LSTM_S** fails altogether, compared with **BERT_S**. This may be due to the residual connections in **BERT**, which enable easy copying of input tokens across layers, while **LSTM** has no such bypass mechanism. Overall, only in few cases, **LSTM_S** outperforms any of **BERT_S**’s layers. This happens on most probing tasks in the case of **SST-2**, where the last layers of **BERT_S** lose practically all linguistic knowledge. Interestingly, in the case of **SST-2** and **Sara**, **LSTM_S** outperforms **BERT_S** on **BigramShift** and **CoordinationInversion** – the two probing tasks that require sensitivity to word and clause order. It may be due to the recurrent nature of **LSTM_S** which makes such order sensitivity an inherent property of the model. The recurrent student is also more sensitive to overall input length.

Relative to the probing results of the embeddings taken before knowledge distillation (see “embed.” in Fig. 6.2), the students sometimes perform better and other times worse. In the morphology-oriented tasks which use a lot of embedding knowledge (**Tense**, **SubjNumber**, **ObjNumber**), **LSTM_S** understandably performs worse than the embeddings and similar to the last layers of **BERT_S**. The embeddings from the trained **BERT_S** achieve mostly comparable probing results, signalling that they did not change much during the student training (in line with the argument that the top layers change the most).

Localising different linguistic knowledge is possible in **BERT_S**⁴. Many skills are, at the first sight, concentrated in and before for first layer (**Length**, **WordContent**, **Tense**, **SubjNumber**, **ObjNumber**), but in most cases this is likely an artifact of the embeddings being the initial source of this knowledge. In other cases, knowledge in **BERT_S** is

⁴Because I probed only one layer of **LSTM_S**, there, localising knowledge is not possible.

clearly concentrated in higher layers (for CoLA on tasks `TopConstituents`, `BigramShift`, `SubjNumber`, `ObjNumber`, `CoordinationInversion`). Interesting is the syntactic task `TopConstituents`, where going from the “E” layer to layer 1 in all BERT_S students increases the probing score significantly. In other words, the trained embeddings contain the necessary knowledge, but at least one model layer is required to aggregate and combine the knowledge across input tokens into sentence-level structural knowledge needed for the task.

6.1.3 What can students learn from scratch

In the previous section, I observed how language knowledge enters student models from the provided pre-trained embeddings. Naturally, such “intruder” knowledge makes it difficult to interpret probing scores of students. In particular, does a high score mean that the student acquired the skill in question during training, or was the knowledge merely present in the injected embeddings?

To control for the initial knowledge present in a student, I probe students trained from scratch, i.e. with embeddings initialised randomly the same way as the rest of the models. In general, the probing results do not differ much from those of students initialised with pre-trained embeddings. In particular, the differences are very small for LSTM_S, which is understandable – I probe the model after its last LSTM layer, whereas injecting pre-trained knowledge tends to affect the first layers. For BERT_S, the variant initialised from scratch often performs slightly worse, which is again understandable given the lack of initial knowledge.

In [Fig. 6.3](#) I show the most striking differences (for full results see [Appendix B](#)). Namely, for BERT_S trained on CoLA and SST-2, on the three probing tasks which I previously described as morphology-oriented and to a great extent relying on embedding knowledge: `Tense`, `SubjNumber`, and `ObjNumber`. Without pre-trained embeddings, all layers of the CoLA BERT_S perform much worse on these probing tasks. While these linguistic skills are arguably important for a task like CoLA, it is clear that the student is unable to learn them well without the embedding knowledge provided beforehand. In the case of the SST-2 student, the original probing scores show progressive decay of performance from earlier to later layers. This, as I previously discussed, can be attributed to the relevant knowledge propagating from the pre-trained embeddings up through the model during training. When the pre-trained knowledge is initially absent, as [Fig. 6.3](#) shows, the decay effect is weak or not present at all. Instead, all layers alike achieve low probing scores. This is a strong signal that the morphology-oriented skills were not needed for the Sara task, and so BERT_S did not learn them well.

All in all, this simple experiment showed that probing results should be interpreted very carefully. In particular, if a model achieves a high probing score, it may be due to different reasons:

- The model as such was pre-trained, with language knowledge present before the task-specific training (like in the case of BERT_T).

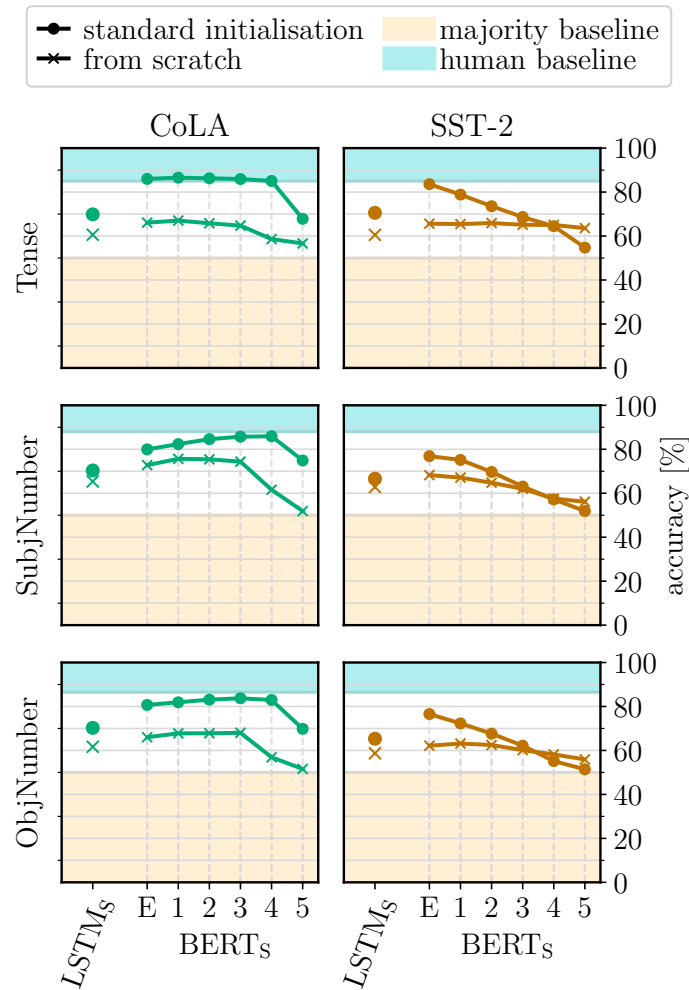


Figure 6.3. Probing results for selected downstream and probing tasks, comparing studentss initialised in the standard way (with embeddings from word2vec) with students initialised randomly and trained from scratch. Bounding the expected model performance from below and from above are again the majority-class baseline and the human performance baseline, same as in Fig. 6.1.

- The model may have received the knowledge (or a large part of it) when its embedding layer was initialised with trained parameters.
- The knowledge may be needed for the task the model is trained to do, and the model actively acquired the knowledge from the data during its training.

6.2 Analysing the models' predictions

This will involve for each downstream task:

1. gathering dev-set predictions made by the two best students and the teacher
2. looking at the mistakes made by each model (most confident mistakes and least confident mistakes first, maybe taking first 10-20 mistakes from each end), trying

to describe them in intuitive terms if possible

3. comparing the mistakes made by different models, again trying to describe what I observe
4. looking at how confident the predictions were in general for each model (maybe the distribution of confidences?) and if possible, comparing this across models

My aim is to make at least some observations describable in human-understandable terms about how the models compare: how the 2 architecturally different students compare, and how their predicting behaviour compares to that of their teacher. Ideally, I will be also able to form some (even if weak) hypotheses that are further testable using probing tasks.

Chapter 7

Overall discussion and conclusions

Chapter 8

Future work

Bibliography

- Adhikari, A., Ram, A., Tang, R., and Lin, J. (2019). DocBERT: BERT for document classification.
- Adi, Y., Kermany, E., Belinkov, Y., Lavi, O., and Goldberg, Y. (2017). Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. *ICLR*, abs/1608.04207.
- Ba, J. and Caruana, R. (2014). Do deep nets really need to be deep? In *NIPS*.
- Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate.
- Belinkov, Y. and Glass, J. R. (2018). Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72.
- Bucila, C., Caruana, R., and Niculescu-Mizil, A. (2006). Model compression. In *KDD '06*.
- Cheong, R. and Daniel, R. (2019). transformers.zip: Compressing transformers with pruning and quantization.
- Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning, ICML 2008*, page 160–167, New York, NY, USA. Association for Computing Machinery.
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12(null):2493–2537.
- Conneau, A. and Kiela, D. (2018). SentEval: An evaluation toolkit for universal sentence representations. *arXiv*, abs/1803.05449.
- Conneau, A., Kruszewski, G., Lample, G., Barrault, L., and Baroni, M. (2018). What you can cram into a single \mathbb{R}^d vector: Probing sentence embeddings for linguistic properties. In *ACL*.
- Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 3079–3087. Curran Associates, Inc.
- 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*.

- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of deep bidirectional transformers for language understanding.
- Dos Santos, C. and Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78.
- 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.
- Goyal, P., Doll ar, 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.
- Graves, A. (2013). Generating sequences with recurrent neural networks.
- 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 *Advances in neural information processing systems*, 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.
- Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. (2019). TinyBERT: Distilling BERT for natural language understanding.
- Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709.
- Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar. Association for Computational Linguistics.
- Kim, Y. and Rush, A. M. (2016). Sequence-level knowledge distillation. *arXiv*, abs/1606.07947.
- Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.
- Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *NIPS*.

- Kusner, M., Sun, Y., Kolkin, N., and Weinberger, K. (2015). From word embeddings to document distances. In *International conference on machine learning*, pages 957–966.
- Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270.
- Lample, G. and Conneau, A. (2019). Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*.
- Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., and Kang, J. (2019). BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*.
- 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 *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.
- 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?
- Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In Bengio, Y. and LeCun, Y., editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.
- Mikolov, T., Karafiát, M., Burget, L., Černocký, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *INTERSPEECH*.
- Mirzadeh, S., Farajtabar, M., Li, A., and Ghasemzadeh, H. (2019). Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *CoRR*, abs/1902.03393.
- Mukherjee, S. and Awadallah, A. H. (2019). Distilling transformers into simple neural networks with unlabeled transfer data.
- Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML*.
- 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?
- Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. URL [https://s3-us-west-2. amazon-](https://s3-us-west-2.amazonaws.com)

- aws.com/openai-assets/researchcovers/languageunsupervised/language_understanding_paper.pdf*.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.
- Romero, A., Ballas, N., Kahou, S. E., Chassang, A., Gatta, C., and Bengio, Y. (2015). FitNets: Hints for thin deep nets. In *In Proceedings of ICLR*.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA.
- Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *NeurIPS EMC² 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 *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.
- Shi, X., Padhi, I., and Knight, K. (2016). Does string-based neural MT learn source syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1526–1534, Austin, Texas. Association for Computational Linguistics.
- Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps.
- 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*.
- Strobel, H., Gehrmann, S., Behrisch, M., Perer, A., Pfister, H., and Rush, A. M. (2018). Seq2seq-vis: A visual debugging tool for sequence-to-sequence models.
- 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. (2019). VideoBERT: A joint model for video and language representation learning.
- Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.
- Tang, R., Lu, Y., and Lin, J. (2019a). Natural language generation for effective knowledge distillation. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pages 202–208, Hong Kong, China. Association for Computational Linguistics.
- 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*.
- 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 *7th International Conference on Learning Representations, ICLR 2019*.
- 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*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.
- Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.
- Warstadt, A., Singh, A., and Bowman, S. R. (2018). Neural network acceptability judgments. *arXiv*, abs/1805.12471.
- 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., Ąukasz 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.
- Yu, S., Kulkarni, N., Lee, H., and Kim, J. (2018). On-device neural language model based word prediction. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 128–131, Santa Fe, New Mexico. Association for Computational Linguistics.
- Zeiler, M. D. (2012). ADADELTA: An adaptive learning rate method.
- Zeiler, M. D. and Fergus, R. (2013). Visualizing and understanding convolutional networks.

Appendix A

Sara dataset

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 <i>slots</i> , 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

Continued on next page

Table A.1 – *Continued from previous page*

ask_whatspossible	asking Sara about the things she can do/help with	how u can help me
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

Continued on next page

Table A.1 – *Continued from previous page*

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
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

Continued on next page

Table A.1 – <i>Continued from previous page</i>		
thank	thanking Sara	amazing, thanks

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

Appendix B

Probing

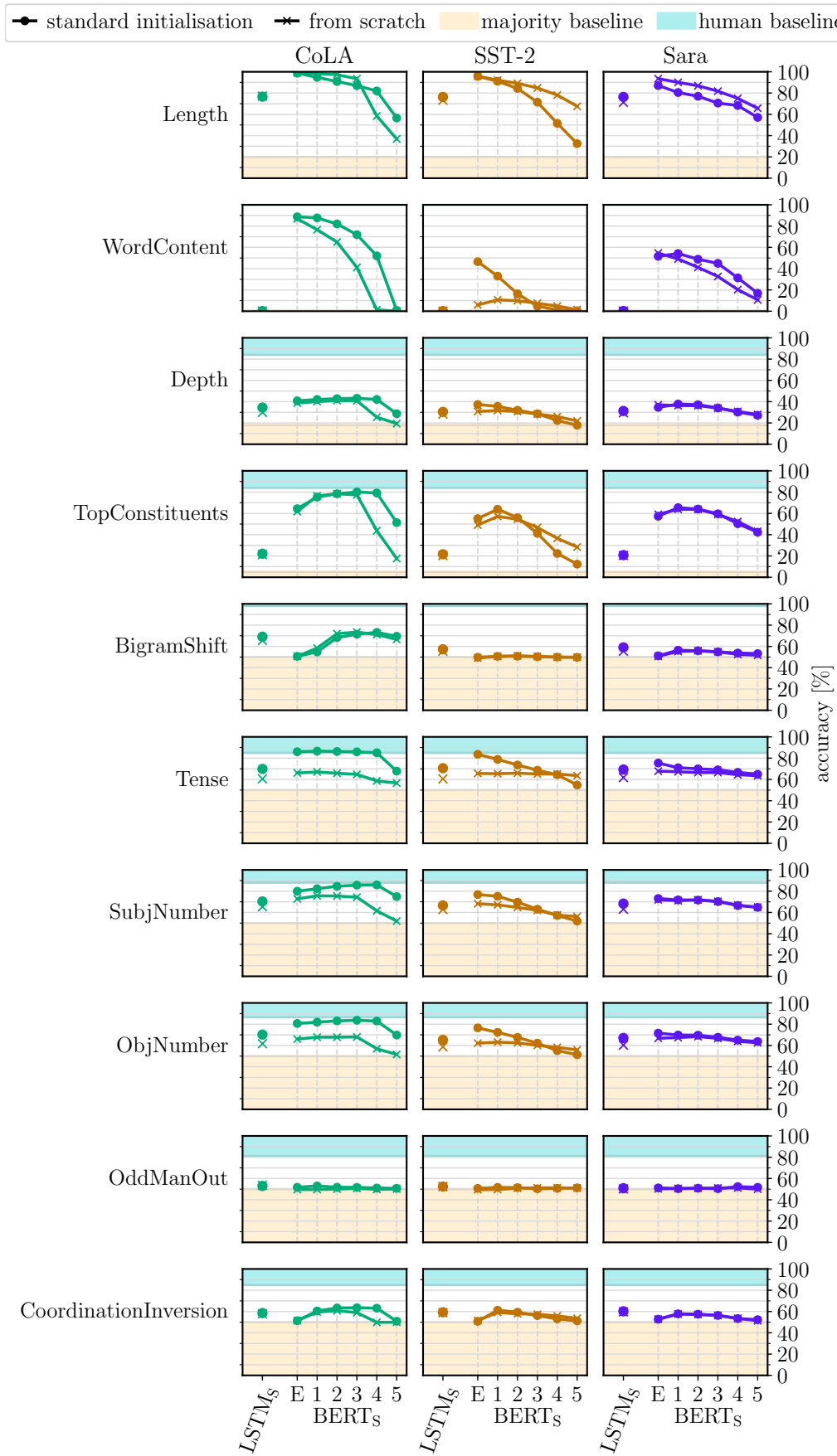


Figure B.1. Probing results comparing studentss initialised in the standard way (with embeddings from word2vec) with students initialised randomly and trained from scratch. Bounding the expected model performance from below and from above are again the majority-class baseline and the human performance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