Teacher-student knowledge distillation from BERT for sentence classifiers

Sam Sučík

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

2020

Abstract

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 . . . 8
1.3 Contributions . . . 8

2 Background 9
2.1 NLP before Transformers . . . 9
2.2 Transformer models and BERT . . . 12
2.3 Teacher-student knowledge distillation . . . 14
2.3.1 Brief history of knowledge distillation . . . 14
2.3.2 Knowledge distillation in NLP . . . 15
2.4 Analysing and understanding NLP models . . . 16

3 Datasets 19
3.1 Downstream tasks . . . 19
3.2 Data augmentation for generating large transfer sets . . . 20
3.3 Probing tasks . . . 21

4 Implementation 23
4.1 Existing implementations used . . . 23
4.2 System overview . . . 23

5 Student tuning 25
5.1 Choosing η . . . 26
5.2 Choosing learning rate annealing . . . 26
5.3 Choosing batch size . . . 26
5.4 Choosing embedding type and mode . . . 26
5.5 Choosing task-specific parameters . . . 27
5.5.1 CoLA . . . 27
5.5.2 SST-2 . . . 28
5.5.3 Sara . . . 28

6 Analysing the models 29
6.1 Analysing the models' predictions . . . 29
6.2 Probing the models for linguistic knowledge . . . 29

7 Overall discussion and conclusions 31

8 Future work 33

9 Plan for the rest of the semester 35

Bibliography 37

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., 2019; 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., 2019; Tang and Lin, 2019), 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 linguistical 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

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 the 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 a larger neural network, which is trained to predict the next word given several previous words, and the previous words given the current word[2]. Such training can easily leverage large amounts of unlabelled text data and the embeddings learn to capture various properties from a word's morphology to its semantics. Released word2vec embeddings became very popular due to their easy use and performance improvements in many NLP tasks. **TO-DO: Add a simple illustration of learning word2vec.**

---

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

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) and later their improved variant – Long Short-Term Memory neural networks (LSTMs) (Hochreiter and Schmidhuber, 1997) – come into play. Although originally proposed long ago, they became popular in NLP, and in text processing in particular, only later (see e.g. Mikolov et al. (2010) and Graves (2013)). These recurrent encoders 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") is outputted and used make a simple prediction.

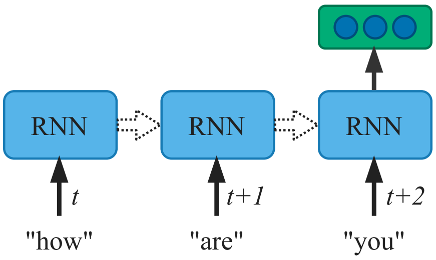

Figure 2.1. A recurrent neural network (RNN) consumes at each timestep one input word. Then, it produces a single vector representation of the inputs.

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 (Kalchbrenner and Blunsom, 2013; Sutskever et al., 2014), see Fig. 2.2. 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.

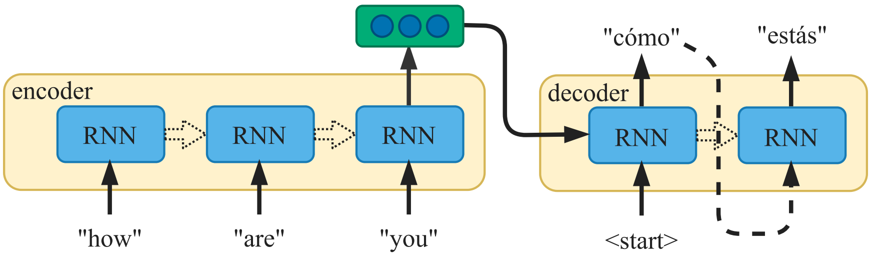

Figure 2.2. An encoder-decoder model for machine translation. Notice how the decoder initially takes as input the special `<start>` token and at later time consumes the previous output word.

---

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

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 parts of the input at each decoder timestep. This is depicted in Fig. 2.3, showing the decoder just about to output the second word ("estás"). The steps (as numbered in the diagram) are:

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 outputted word ("cómo") with the context vector and consuming this information to better decide which word to output next.

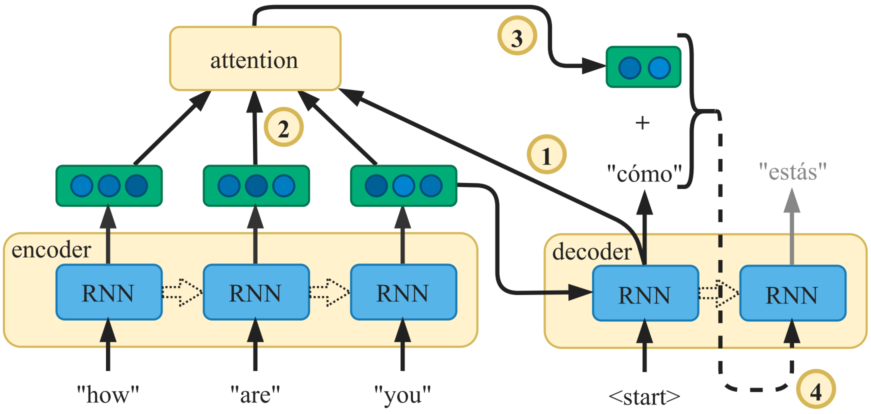

Figure 2.3. An encoder-decoder model for machine translation with added attention mechanism.

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

$$\boldsymbol{q} = \boldsymbol{h_D} W_Q$$

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

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

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

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

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

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

---

[3] My 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.

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

For 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 models and BERT

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 results of this sequentiality, with no room for parallelisation. 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. 2.3, a high-level diagram of the workings of a self-attention layer is shown in Fig. 2.4. 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).

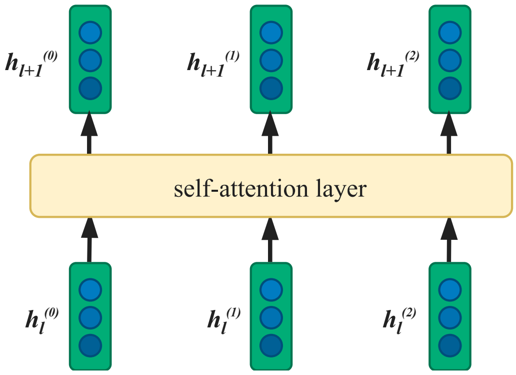

Figure 2.4. A high-level diagram of the application of self-attention in Transformer models. Three hidden states are shown for consistency with the length of the input shown in Fig. 2.3; in general, the input length can vary.

As for the internals of self-attention, the basic principle is very similar to standard attention. Self-attention too is used to focus on and gather relevant information from a sequence of elements, given a query. However, to produce a richer contextual embedding $\boldsymbol{h}_{l+1}^{(i)}$ in layer $l+1$ of the i-th input word, self-attention uses the incoming representation $\boldsymbol{h}_l^{(i)}$ for the query, and considers focusing on all representations in layer l, including $\boldsymbol{h}_l^{(i)}$ itself. Fig. 2.5 shows this in detail for input position $i=0$. Query $\boldsymbol{q}^{(0)}$

is produced and matched with every key in layer l (i.e. $\boldsymbol{k}^{(0)}, \ldots, \boldsymbol{k}^{(2)}$) to produce the attention weights. These weights quantify how relevant each representation $\boldsymbol{h}_l^{(i)}$ is with respect to position $i = 0$. Then, the new contextual embedding $\boldsymbol{h}_{l+1}^{(i)}$ is constructed as a weighted sum of the values $\boldsymbol{v}^{(0)}, \ldots, \boldsymbol{v}^{(2)}$ (same as constructing the context vector in standard attention).

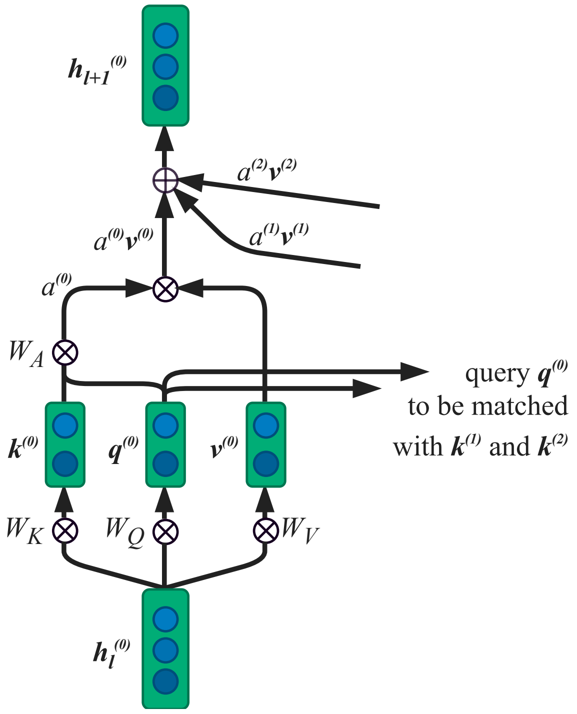

Figure 2.5. The internals of self-attention, illustrated on creating the next-layer hidden representation of the input position $i = 0$, given all representations in the current layer. Note that $\otimes$ stands for multiplication (matrix or scalar), and $\oplus$ denotes summation.

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

Radford et al. (2018) introduced the idea of generative LM pre-training and fine-tuning. This concept helps train much better models even for low-resource tasks with small datasets, by leveraging general language knowledge acquired by the model in the pre-training phase. Publishing pre-trained model instances makes the power of NLP much more accessible to anyone and has become a popular thing to do. (Also mention that subword tokens were used instead of words.)

Devlin et al. (2018) improved the concept by pre-training the model bi-directionally (leading to language modelling based on left *and* right context). They also changed the pre-training to 2 tasks trained at the same time: masked language modelling to learn to understand words, and next sentence prediction to learn to reason about entire sentences (as the actual NLP tasks often require such reasoning). This is is how BERT was born, which then became extremely popular in the community, attracting a lot of work on improving it, analysing its capabilities, extending it to other languages, and even applying it to multi-modal tasks such as video captioning. TO-DO: elaborate more on BERT, also with a schematic picture.

Following the success of BERT, further and often bigger Transformer models started emerging:

- GPT-2 (Radford et al., 2019), a bigger and improved version of GPT
- XLM (Lample and Conneau, 2019) with added introduced cross-lingual pre-training
- Transformer XL (Dai et al., 2019) with better handling of much longer contexts
- and many others

Although the open-sourced powerful pre-trained models were a huge step towards more accessible NLP, the model size meant they couldn't be applied easily outside of research: They were memory-hungry and slow. This inspired another wave of research: compressing the huge, well-performing Transformers (very often BERT) to make them faster and resource-efficient. I will focus on the compression method that so far looks the most effective: knowledge transfer from huge models into smaller ones using teacher-student knowledge distillation.

2.3 Teacher-student knowledge distillation

2.3.1 Brief history of knowledge distillation

The concept of knowledge distillation was introduced by (Bucila et al., 2006) as a way of knowledge transfer from huge models (or ensembles of models) into small ones, with the aim of having smaller (and hence faster) yet well-performing models. The main idea is to train a big neural classifier model also called the *teacher* and then let a smaller neural classifier model (called the *student*) learn from it – by learning to mimic the teacher's behaviour. Hence also the term *teacher-student knowledge distillation*. The mimicking was originally realised by labelling a dataset with the trained teacher model and training the student model on these labels (retrospectively referred to as *hard labels*). The dataset used for training the student model is often called the *transfer dataset*.

Later, Ba and Caruana (2014) introduced the idea of learning from the *soft labels*, i.e. learning to predict the teacher's logits. The idea is that the teacher's knowledge can be transferred to the student in a richer way if the entire logit distribution is utilised (compare with hard labels, when only the information about the maximum element of the logit distribution is utilised). When first proposed, the student's loss function was then mean squared distance between the student's outputs and teacher's logits.

Hinton et al. (2015) proposed a more general approach, addressing the issue of overconfident teachers with very sharp logit distributions, and formulated a new loss for student models: the cross-entropy loss with temperature. The temperature softens the teacher's sharp distribution, allowing the student to learn more rich information from it. This approach is what is today typically referred to as knowledge distillation.

Since 2015, further variants have been proposed, enhancing knowledge distillation in different ways, for example:

- Papamakarios (2015, p. 13) points out that mimicking teacher outputs (e.g. with cross-entropy loss) can be extended to mimicking the derivatives of the loss w.r.t. inputs (i.e. including in the loss function also this term: $\frac{\partial \mathbf{o}_{student}}{\partial \mathbf{x}} - \frac{\partial \mathbf{o}_{teacher}}{\partial \mathbf{x}}$).
- Romero et al. (2015) introduced the idea of learning to match not just the teacher's outputs, but also its intermediate input representations. Huang and Wang (2017) achieved this by learning to align the distributions of neuron selectivity patterns between the teacher's and student's hidden layers. Unlike standard knowledge distillation, this approach no longer works just for classifier models with a softmax output layer (see the standard loss proposed by Hinton et al. (2015)).
- Sau and Balasubramanian (2016) show that learning can be more effective when noise is added to the teacher logits.
- Mirzadeh et al. (2019) show that a knowledge distillation performs poorly from large teachers into very small students and alleviate this limitation by multi-stage distillation: first distilling knowledge into an intermediate-size "teacher assistant" and learning the student from the assistant model.

2.3.2 Knowledge distillation in NLP

Practically all the so far mentioned research in knowledge distillation was done in the domain of image processing. This comes as no surprise: It was image processing that was benefitting the most from the resurgence of deep learning. Ever since the AlexNet (Krizhevsky et al., 2012), bigger and bigger models were proposed, simultaneously driving the research in model compression so as to make the models usable in practice.

In natural language processing, research on knowledge distillation was rare for a long time. One notable work was the adaptation of distillation for sequence-to-sequence machine translation models – whose outputs are no longer simple classification scores – by Kim and Rush (2016). Another pioneering study compressed a recurrent neural language model for use on mobile devices (Yu et al., 2018).

However, the real need for model compression started very recently when huge Transformer models were introduced. This follows on from the main limitation of the otherwise very accessible pre-trained models: being huge and slow.

When distilling knowledge from big, pre-trained Transformer models, the main decision is whether to distil before or after fine-tuning on a concrete downstream task. Each option has its pros and cons. In the first scenario, the steps are: 1) distilling into a small student, 2) fine-tuning the student on any downstream task. One advantage is that the distillation is done only once and fine-tuning the student can be fast (due to the student's size). Since the distillation can be done on the same data that the teacher was pre-trained on – large corpora of unlabelled data, one does not have to worry about not having enough data. One possible downside is that the amount of general language knowledge contained in the pre-trained teacher will be too much to contain within a small student, hence requiring students that are themselves considerably large (and slow). Sanh et al. (2019) took this approach and while their student model is very successfully fine-tuned to a wide range of tasks, it smaller than the teacher (BERT base) only by 40%, still having 66M parameters.

In the second scenario, the steps are: 1) fine-tuning the pre-trained teacher on a downstream task, 2) distilling the teacher into a small student. In this case, the downstream task-specific language knowledge is likely to be much smaller than the teacher's overall pre-trained language knowledge, meaning that a much smaller student can still contain all important knowledge from the fine-tuned teacher. However, with this approach, the teacher fine-tuning and distillation has to be done separately every downstream task, which can be resource-intensive. Even more importantly, the distillation procedure can suffer from lack of data, since the distillation is now being done only with the downstream task dataset, which will often be small and has to be augmented. Various ways of augmenting small datasets to increase the amount of transfer data have been proposed, with mixed success. Mukherjee and Awadallah (2019) use additional unlabelled in-domain sentences with labels generated by the teacher – an approach that only works if such in-domain sentences are available. Tang et al. (2019) propose augmentation based on simple, rule-based perturbation of existing sentences from the downstream task data. Finally, Jiao et al. (2019) and Tang and Lin (2019) use big Transformer models generatively to create new sentences. In the first case, BERT is repeatedly used to choose a suitable replacement for an existing word in a given sentence, eventually leading to a new sentence with many words changed for different ones. In the second case, a GPT-2 model is fine-tuned on the downstream task with a language model objective, and a new sentence is generated by repeatedly predicting the next token, conditioned on the sequence generated so far.

In this work, I adopt the approach of Tang and Lin (2019) as I view it as the most promising one so far. However, while they use only bidirectional LSTM students, I also experiment with a smaller version of BERT, similarly to Jiao et al. (2019). For detailed description of the system see Chapter 4.

2.4 Analysing and understanding NLP models

NNs are black boxes and (not) understanding the models is a serious issue. performance is typically more important than transparence, but recently the demand for explainable AI (XAI) has been increasing. additionally, understanding is opportunity for further improvements of the mdels and techniques.

in image processing, interpretability is easy thanks to visualising things. (somewhat similarly music.) notable works: maximising activation, visualising neurons' output for given input, maximum activation samples. in NLP, interpreting is more difficult, and also comes with a delay after image processing – same with the big model hype and compression hype. Belinkov and Glass (2018) give nice overview of work done up until 2018. they point out that many methods for analysing and interpreting models are adapted from image processing, especially visualising activations of single neurons on specific input examples. in attentional seq2seq models, the attention maps can be visualised to show soft alignments between input and output sequences. however, these methods taken from image processing are mostly qualitative, not suited for comparing models.

more quantitative and NLP-specific are the approaches that look at kinds of linguistic knowledge present in a model's internal representations of input. most often, this means extracting activations for a set of inputs and trying to predict properties of the input from the activations by using a simple predictor model. if the prediction works well, then the activations must've contained linguistic knowledge relevant for the predicted property. since first proposed by Shi et al. (2016), this approach has been used repeatedly to explore various forms of representations. the first work was looking at how well NMT systems capture syntactic properties of input. later, Adi et al. (2017) used simpler artificial prediction tasks (sentence length, word content and word order) to better understand sentence embeddings produced by recurrent encoders. more recently, Conneau et al. (2018) curated a set of 10 probing tasks ranging from surface properties through syntactic to semantic ones, and compared different recurrent and convolutional sentence encoders (and many useful baseline models) in terms of the linguistic knowledge in their sentence representations. focusing specifically on Transformer models, Tenney et al. (2019b) propose a set of *edge probing* tasks which examine how much contextual knowledge about the entire sentence is captured within the representation of a single word. the tasks mimic the typical steps of a standard NLP pipeline – from POS tagging to identifying dependencies and entities to semantic role labelling. Tenney et al. (2019a) were able to localise the layers of BERT that are the most important for each of the tasks, showing that the different linguistic capabilities are ordered within the model in the typical NLP pipeline order: from simple POS tagging in the earlier layers to the most complex semantic tasks in the last layers.

while the abovementioned methods for analysing models provide valuable insights, they are incomplete. for one, they merely help us describe in intuitive terms the kinds of internal knowledge/specialisation observed in the models. in their overview of interpretability of machine learning, Gilpin et al. (2018) call this level of model understanding *interpretability* – comprehending what a model does. however, 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 for analysing neural NLP models achieve only interpretability because they enable us to describe but not explain (especially in terms of causality) the internals and decisions of the models.

In this work, I also attempt to mainly *interpret* the student and teacher models. I adopt two approaches:

1. analysing the mistakes the models make on the downstream task they were trained to do, including how confident the correct and incorrect predictions are
2. probing the models using the probing tasks curated by Conneau et al. (2018)

By comparing the findings between models trained on different downstream tasks or with different architectures, I try to characterise each task in terms of the linguistic capabilities it utilises. Further, I describe how different student model architectures influence how linguistic knowledge is distilled from a teacher and stored in the student, and what the effect on the student's confidence is. Finally, I try to relate the observed effects to the method of knowledge distillation itself.

Chapter 3

Datasets

3.1 Downstream tasks

Since the BERT model I use as teacher is already pre-trained on large unlabelled corpora (for details see the original work by Devlin et al. (2018)), it can be fine-tuned well even on small downstream datasets.

Today, perhaps the most popular collection of challenging NLP tasks (challenging by the nature of the tasks and by the relatively small dataset size) is the GLUE benchmark (Wang et al., 2018). This collection comprises 11 tasks from semantic analysis to detecting textual similarity to natural language inference, framed as single sentence or sentence pair classification. Each task features a specific scoring metric (such as accuracy or F1), publicly available labelled training and development datasets, and a testing dataset whose labels have not been released. The test-set score accumulated over all tasks forms the basis for the popular GLUE leaderboard[1].

For simplicity, I use only the single-sentence classification tasks of the collection – the Corpus of Linguistic Acceptability (CoLA) and the Stanford Sentiment Treebank in its two-way classification format (SST-2).

- The CoLA task features 8.5k training sentences, 1k evaluation and 1k testing sentences. The (hand-crafted) sentences are collected from various linguistic literature and represent examples of acceptable and unacceptable English, making the task a binary classification. The scoring metric is Matthew's Correlation Coefficient (MCC, introduced by Matthews (1975)). This task is rather challenging since a given sentence can be perfectly grammatical and still not be acceptable English. As a non-native speaker, I myself struggle with some of the examples.
- The SST-2 task has considerably more data as it is easier to collect: 67k training, 9k evaluation and 18k testing sentences, extracted from movie reviews and labelled as positive or negative sentiment. The scoring metric is accuracy. The task is easier than CoLA and best models in the GLUE leaderboard achieve over 97% test-set score.

---

[1] `https://gluebenchmark.com/leaderboard`

As the third task, I use an intent classification dataset collected by Rasa, a company building open-source tools for conversational AI[2]. The dataset is named Sara, after the company's own chatbot[3] which provides various information about Rasa and its products to the visitors of the company's website[4]. The dataset comprises 4.8k examples overall, split between 1k testing portion and 3.8k training portion. Each example is a human-generated utterance and has been manually labelled with one of 57 intents, helping the Sara chatbot learn to detect what a human is talking to it about. Originally, the dataset was initialised by Rasa employees talking to Sara. Later, many more examples were collected by deploying the chatbot and letting site visitors talk to it. The 57 intent types were designed by Rasa, starting with a smaller number of broader intents and refining the set after observing real questions humans were asking the chatbot. **TO-DO: verify this is true; my knowledge of the history of Sara is limited.** The main scoring metric is the multi-class micro-averaged F1 score.

I modified the Sara dataset by splitting the training portion into 2.8k training and 1k evaluation sentences (while keeping both sets class-balanced). I also anonymised the dataset by replacing all sensitive information with generic tokens. Namely, I replaced people's names and surnames with a single token `__PERSON_NAME__` and e-mail addresses with `__EMAIL_ADDRESS__`.

To stay consistent with the GLUE datasets for which test-set labels are not publicly available, I do not use the test portion of any of them when analysing the mistakes made by different models. Instead, I analyse the predictions on the evaluation set. This can be viewed as sheding light on the kind of model qualities that are actually optimised when choosing model parameters on this set.

3.2 Data augmentation for generating large transfer sets

As Tang et al. (2019) demonstrate, taking just the training sentences of a downstream task as the transfer set for knowledge distillation is not enough; especially for challenging tasks like CoLA where the training set comprises only several thousands of sentences (for results, see Table 1 in Tang and Lin (2019)). I adopt the approach that they found to work the best: Augmenting the training portion with additional sentences generated by a GPT-2 model (Radford et al., 2019). The concrete steps for this augmentation are:

1. Fine-tune the pre-trained GPT-2 model (the 345M-parameter version) on the training sentences for 1 epoch with the language-modelling objective, i.e. learning to predict the next subword token of a sentence given the true sequence of tokens so far.
2. Sample a large number of prefixes (subword tokens) from the observed distribution of sentence-initial subword tokens in the training sentences.

---

[2] For transparency: I did a 3-month internship as Machine Learning researcher with Rasa in the summer of 2019.

[3] Demonstrated at https://github.com/RasaHQ/rasa-demo

[4] https://rasa.com/

3. For each sampled prefix, generate a sentence starting with it: by sampling from the predicted next-token distribution of the GPT-2 model, starting with just the prefix and stopping when the special end-of-sentence token is generated or the desired maximum sequence length is reached (in this case 128 tokens).
4. Add the generated sentences to the original training sentences and use as the transfer set, i.e. append with teacher-generated logits and use to train the students.

For consistency, I used the same number of augmentation sentences as Tang et al. (2019): 800k for each downstream task. Hence, the transfer set comprises 808.5k sentences for CoLA, 867k sentences for SST-2, and 802.8k sentences for Sara.

3.3 Probing tasks

For exploring the linguistic knowledge in the different teacher and student models, I use the probing task collection curated by Conneau et al. (2018). The datasets, along with tools for applying probing to neural models, are publicly available as part of the SentEval toolkit Conneau and Kiela (2018) for evaluating sentence representations. The collection contains 10 tasks, each having 100k training, 10k evaluation and 10k testing sentences. These are used for training the probing classifier, choosing its hyperparameters, and computing the task-specific score, respectively.

Each of the 10 probing tasks – grouped into three broad categories – focuses on a different linguistic capability (for more details, see the original paper of Conneau et al. (2018)):

Surface information:

- **Length** is about recovering the length of the sentence. The actual sentence lengths are 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 created, and sentences were sampled such that each contains exactly one of these words. The task is therefore 1000-way classification.

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 recognising sentences in which the order of two randomly chosen adjacent words has been swapped (binary classification).
- **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 sequences and the “other” option.

Semantic information:

- **Tense** is binary classification, identifying the tense (present/past) of the main verb of the sentence (the verb in the main clause).
- **SubjNumber** is about determining the number (singular/plural) of the sentence's subject, framed as binary classification.
- **ObjNumber** is the same as SubjNumber, applied to the direct object of a sentence.
- **OddManOut** is binary classification, detecting whether a random word (verb or noun) in a sentence was replaced with a different one (verb or noun) or not. 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.
- **CoordinationInversion** works with sentences that contain two coordinate clauses whose order may have been artificially swapped. The binary classification is then about detecting which sentences are intact.

Considering the findings of Tenney et al. (2019a) about the NLP pipeline steps being implicitly captured inside BERT, choosing their approach to probing (together with the collection of tasks they curated) was an attractive option. However, the approach explores single-word representations, focuses primarily on more complex probing tasks (e.g. entity recognition, natural language inference, semantic roles) and the tasks are not open-sourced. In this sense, the SentEval toolkit is preferred as it was built for analysing sentence representations, the simpler surface and syntactic tasks have a better coverage, and the datasets are publicly available. In the future, however, the two collections of probing tasks could be used in combination.

Chapter 4

Implementation

4.1 Existing implementations used

For finetuning BERT, I used `transformers` (Wolf et al., 2019)[1] – a popular open-source PyTorch library curating and presenting many successful Transformer-based models in a unified way. My implementation of the distillation process is heavily based on the implementation accompanying DistilBERT (Sanh et al., 2019), which is today part of `transformers`[2].

The implementation of the BiLSTM student is based on the codebase accompanying the work of Tang and Lin (2019)[3] (which was originally built using an early version of the `transformers` library). The same codebase was used without many changes for fine-tuning the GPT-model and creating the augmentation sentences.

Finally, for probing, I adapted the SentEval toolkit (Conneau and Kiela, 2018) to suit my needs.

In terms of implementation, the most of my own work is in having integrated the different tools. In particular, I have combined the codebases of Sanh et al. (2019) and Tang and Lin (2019) and adapted the combination to offer richer options (e.g. initialising the student's embedding layers with trained word or wordpiece embeddings), to work with both BERT and BiLSTM students and to be more easily extendable to other architectures.

4.2 System overview

Following Tang and Lin (2019), I used the large, case-insensitive (uncased) pre-trained BERT (340M params) as the teacher and finetuned it for 3 epochs using Adam optimizer with $\eta = 5e-5$ with linear warm-up over the first 10% of training steps, followed by linear decay, and with batch size $B = 36$. I will refer to this model as $BERT_T$. Somewhat surprisingly, while the teacher converged within 3 epochs for CoLA and SST-2 (in terms of the dev-set score), for Sara, it converged much more slowly. I empirically chose 10 epochs as the smallest number after which $BERT_T$ would sufficiently converge on the Sara dataset.

---

[1] `https://github.com/huggingface/transformers`, accessed January 30, 2019

[2] `https://github.com/huggingface/transformers/tree/master/examples/distillation`, accessed January 30, 2019

[3] `https://github.com/castorini/d-bert`, accessed January 30, 2019

As students, I used:

- a bi-directional LSTM same as the one used by Tang and Lin (2019): with one LSTM layer with 300 units, followed by a linear layer with 400 units and ReLU activation function, with a softmax classifier on top, with 2.41M non-embedding trainable parameters. I will refer to this model as $LSTM_S$.
- a smaller version of the teacher BERT model, with all dimensions down-scaled 5x to roughly match the number of trainable non-embedding parameters: with 5 transformer layers, hidden dimension of 204, intermediate layers of size 750, and 3 attentional heads (amounting to 2.42M parameters). I will refer to this model as $BERT_S$.

$LSTM_S$ was trained to minimise the mean squared loss (MSE) between the teacher-generated logits and student's logits. While the cross-entropy loss with configurable temperature is more popular and in essence a generalisation of the MSE loss (see Hinton et al. (2015)), Tang and Lin (2019) report that the MSE loss led to slightly better performance in their experiments. For $BERT_S$, I use the traditional cross-entropy loss with temeperature set to $T = 3$ following the findings of Tsai et al. (2019) who experimented with different temperatures (note that $T = 3$ was used as the default also in the experiments of Tang and Lin (2019)).

Following preliminary experiments, I limited the number of training epochs to 30 for $LSTM_S$ (same budget was used by Tang and Lin (2019)) and to 60 for $BERT_S$ as it was converging much slower than the recurrent student. I stick to this training budget throughout my experiments.

some overall visualisation of all the components and how they interact

Chapter 5

Student tuning

Importantly, extensive tuning of the students' hyperparameters is certainly not the main objective of this work. However, my aim is to carry out an analysis of the teacher and student models and the distillation process. For such analysis to be of some value, it should be done using well-performing models which themselves are of practical value.

I tune a number of hyperparameters for the 2.4M-parameter students using the CoLA task. Afterwards, I tune a smaller number of parameters – which are the most likely to be task-dependent – separately for each downstream task, trying to obtain $LSTM_S$ and $BERT_S$ that would achieve 90% of the teacher's performance while being as small as possible. These student are then used for further analysis.

For both students, I first tune the following, in this order (for more details, see the next sections):

1. learning rate η
2. learning rate scheduling, more concretely the warm-up and decay of η
3. batch size B
4. embedding type (wordpiece vs word) and mode

Afterwards, I tune on each task separately the embedding type and the student size.

The starting configuration (which I change as I tune the different parameters) is:

- $BERT_S$: Adam with $\eta = 5e-5$, $B = 256$, learning rate warm-up over the first 10 epochs, followed by linear decay, and wordpiece embedding layer
- $LSTM_S$: Adadelta with $\eta = 1.0$, $B = 50$, no learning rate annealing, and word embedding layer in the multichannel mode, using word2vec

Initially, $LSTM_S$ had its embedding layer initialised with word2vec embeddings, but $BERT_S$ was initialised entirely from scratch. Naturally, this was posing a disadvantage for $BERT_S$ as it was starting with no knowledge. Hence, I added the option to initialise the embedding layer with wordpiece embeddings from the corresponding finetuned $BERT_T$. I report results of parameter tuning for both of these variants of $BERT_S$.

5.1 Choosing η

Because Tang and Lin (2019) report not tuning their hyperparameters, I acknowledge their parameter choices but challenge many of them. In particular, they use Adadelta with $\eta = 1.0$ and $\rho = 0.95$ as the learning algorithm. I attempt to use the Adam algorithm as it is a generalised, improved version of Adadelta. Hence, for both students, I compare values of η from [5e-3, 1.5e-3, 5e-4, 1.5e-4, 5e-5, 1.5e-5, 5e-6] for the Adam optimizer, with betas fixed to 0.9 and 0.98.

TO-DO: add graphs

The best value was found to be $\eta = 5e-4$ for $LSTM_S$ and bth variants of $BERT_S$. For $BLSTM_S$, I abandoned the Adadelta optimizer as it performed worse than Adam.

5.2 Choosing learning rate annealing

I tried linear warm-up of η (starting from 0) over [0, 5, 10, 15, 20] epochs E_{WP} for $BERT_S$ and over [0, 5, 10, 15] epochs for $LSTM_S$.

TO-DO: add graphs

For $BERT_S$ initialised from scratch, $E_{WP} = 20$ and no decay worked the best. For the other students, decay was improving the performance slightly, with the best E_{WP} being 0 for $LSTM_S$ and 15 for $BERT_S$

5.3 Choosing batch size

Note that for $LSTM_S$ this was done prior to exploring the learnign rate annealing, as my aim was to merely verify that $B = 50$ reported by Tang and Lin (2019) was the best value. For $BERT_S$, I had much less intuition about suitable batch sizes. The explored values were [32, 64, 128, 256, 512] for $BERT_S$ and [32, 50, 128, 256, 512] for $LSTM_S$.

TO-DO: add graphs

Interestingly, while $LSTM_S$ preferred the smalles batch size $B = 32$, $BERT_S$ preferred much bigger batches: 256 when initialised from scratch and 128 with wordpiece embeddings initialised from $BERT_T$.

5.4 Choosing embedding type and mode

After observing that the best $LSTM_S$ consistently outperforms its counterpart $BERT_S$, I turned to exploring the effect of pre-trained knowledge present in the student before distillation. In $LSTM_S$, this lies in the word2vec embeddings. Additionally, the embedding layer is used in the *multichannel mode*: two parallel embedding layers are initialised from word2vec, with one being frozen during distillation and the other one allowed to be finetuned. In $BERT_S$, there is no pre-trained knowledge or there is the knowledge from the teacher's wordpiece embeddings (dimensionality reduced by a trainable linear layer to match the smaller hidden size of $BERT_S$.

I was interested to see whether the effect of different type of embeddings – word2vec pre-trained on general language modelling vs wordpiece pretrained on language modelling and finetuned on the particular downstream task – could explain some of the performance gap between $BERT_S$ and $LSTM_S$. Hence, I tried each of the 2 students with each of the 2 embedding types, combined with each of the following 2 modes: the multichannel mode described above, and the simple (*non-static*) mode where only one embedding instance is created, which is unfrozen and allowed to further train during distillation.

TO-DO: add graphs

Turns out $LSTM_S$ strongly prefers word embeddings in the multichannel mode, while $BERT_S$ is less decisive, with only slight preference for word embeddings and the multichannel mode. This also shows that the different embedding properties don't explain the worse performance of $BERT_S$.

5.5 Choosing task-specific parameters

For each dataset, I first tried different embedding types and modes, and then for the best-performing combination I tried up-scaling the students to get as close to 90% of $BERT_T$'s performance as possible, while keeping the size of each student reasonably smaller than the teacher.

5.5.1 CoLA

Here I had the optimal embedding type/mode already.

I tried expanding $LSTM_S$ by increasing the number of LSTM layers from 1 to [2, 3, 4, 5] and the width of the LSTM and fully-connected layers from the original 300 and 400 by 2x, 3x, 4x and 5x. In case of $BERT_S$, I tried increasing the number of layers from the original 5 to 10 and 15, and the other parameters (# of attentional heads, hidden and intermediate size) by 2x, 3x and 4x.

TO-DO: add graphs

For $LSTM_S$, the results are indecisive, without clear correlation between student size and performance. The best student was 2x deeper and 2x wider than the initial configuration.

For $BERT_S$, I soon observed that bigger models started dying after the first few epochs. TO-DO: add diagnostic graphs. This dying surfaced as the dev-set score flattening out and then decreasing towards 0, at which point the gradients in the model would explode. I identified the cause of this to be too big learning rate (thanks to the warm-up, the model was experiencing η starting from 0 and increasing towards the target value). Hence, by comparing the graphs for the linearly increasing η and the dev-set performance, I tried to manually identify η values as big as possible that were still enabling the model to improve fast (without making it plateau or die). These values were in the range from the originally chosen 5e-4 down to 8e-5 (generally lower η for bigger models). After this modification, all students trained normally, without dying. TO-DO: add graphs and conclusions once all of these have finished training.

5.5.2 SST-2

TO-DO: run experiments and report

5.5.3 Sara

TO-DO: run experiments and report

Chapter 6

Analysing the models

6.1 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 then 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.

6.2 Probing the models for linguistic knowledge

Here, I will for each downstream task:

- probe each of the 3 models in different layers (e.g. the 24-layer $BERT_T$ in layers 1, 6, 12, 18, 24) to see how much linguistic knowledge and where is present in each model
- probe also the pretrained BERT (not finetund on any downstream task) and look at how knowledge is preserved/thrown away when the teacher gets finetuned vs when students with little prior knowledge are trained during distillation

- compare how knowledge is present in the two architecturally different students, try to relate this to other differences noticed when analysing the students' predictions

For now, I have one graph summarising the results of probing just the teacher models, see Fig. 6.1.

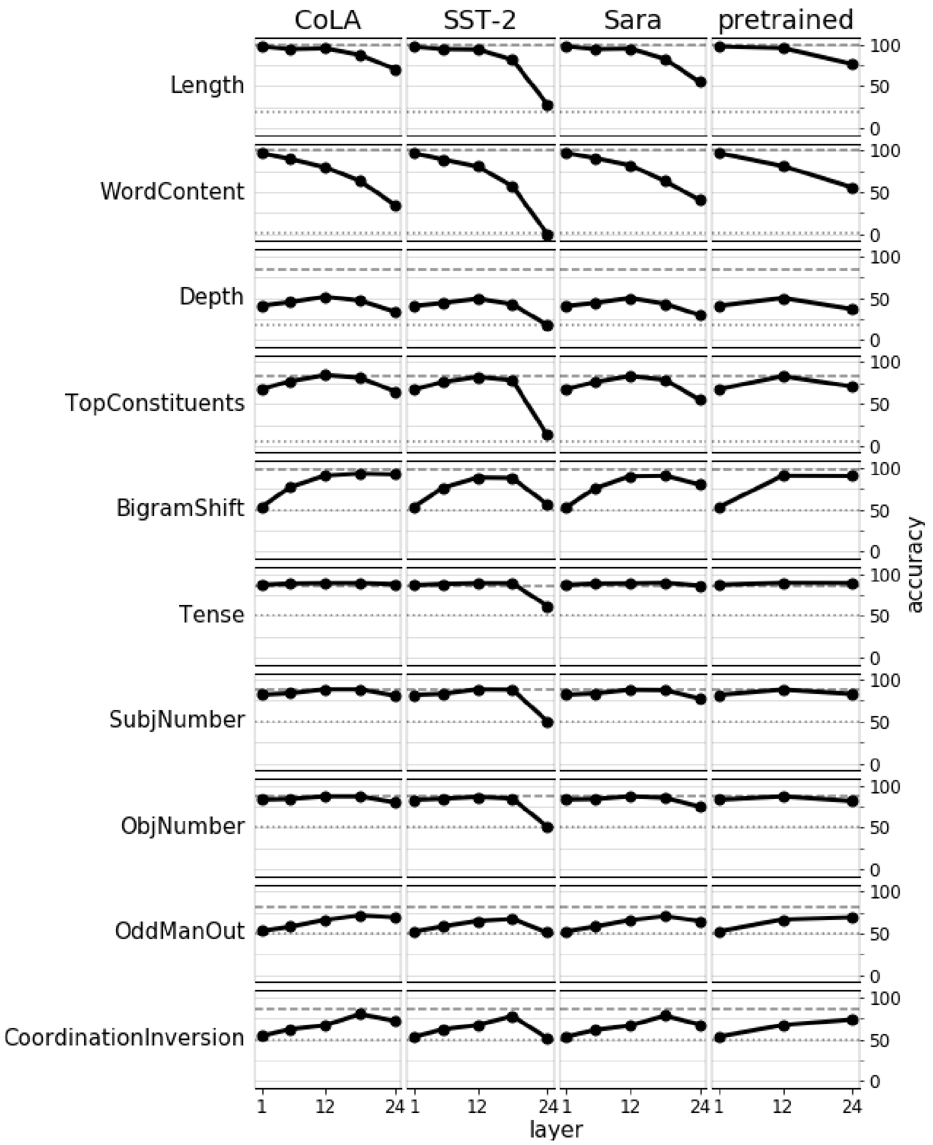

Figure 6.1. Probing the 3 finetuned teachers and the pretrained one. The dotted line shows baseline accuracy levels achieved by majority-class guessing, the dashed line shows human performance.

Chapter 7

Overall discussion and conclusions

Chapter 8

Future work

Chapter 9

Plan for the rest of the semester

Work finished:

- implementing everything related to distillation (incl. teacher finetuning and generating augmentation data with GPT-2)
- creating augmented transfer sets and finetuning the BERT teacher for each task
- exploring the model parameters shared across tasks (i.e. doesn't include embedding type and student size)
- implementing everything needed for probing
- implementing everything needed for gathering predictions for mistake/confidence analysis

Work to be done:

- now-Feb 10: finishing size exploration for SST-2 and Sara (ending up with best students for each downstream task)
- Feb 11-23: analysing the predictions of all models (includes Inovative Learning Week Feb 17-21 when I won't get terribly much done)
- Feb 11-16: probing the best students
- Feb 24-Mar 1: putting together overall analysis and conclusions
- Mar 2-Apr 2: buffer time and writing the report (a lot of coursework in this period as well)

Bibliography

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 160167, 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):24932537.

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 $&!#* vector: Probing sentence embeddings for linguistic properties. In *ACL*.

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.

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.

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.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):17351780.

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. and Rush, A. M. (2016). Sequence-level knowledge distillation. *arXiv*, abs/1606.07947.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*.

Lample, G. and Conneau, A. (2019). Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*.

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.

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

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. 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 318362. 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.

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

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.

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

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. and Lin, J. (2019). Natural language generation for effective knowledge distillation.

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

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.

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.