Knowledge Distillation for Swedish NER models: A Search for Performance and Efficiency

Lovisa Hagström
Chalmers University of Technology
Sweden
lovhag@chalmers.se

Richard Johansson
University of Gothenburg
Sweden
richard.johansson@cse.gu.se

Abstract

The current recipe for better model performance within NLP is to increase model size and training data. While it gives us models with increasingly impressive results, it also makes it more difficult to train and deploy state-of-the-art models for NLP due to increasing computational costs. Model compression is a field of research that aims to alleviate this problem. The field encompasses different methods that aim to preserve the performance of a model while decreasing the size of it. One such method is knowledge distillation. In this article, we investigate the effect of knowledge distillation for named entity recognition models in Swedish. We show that while some sequence tagging models benefit from knowledge distillation, not all models do. This prompts us to ask questions about in which situations and for which models knowledge distillation is beneficial. We also reason about the effect of knowledge distillation on computational costs.

1 Introduction

Currently, most research that pushes the boundary for state-of-the-art performance within natural language processing involves the increase of number of model parameters as well as the computations needed for training (Devlin et al., 2019; Radford et al., 2019; Brown et al., 2020). The trend seems to be that the larger the model, the better the performance. As noted by Strubell et al. (2019) these state-of-the-art models require significant computational resources during training as well as deployment. While it certainly is a good thing that state-of-the-art performance within NLP is continuously improving, there is work to be done on model efficiency. More efficient models are needed both for the sake of the environment and for the sake of equal research opportunities. Here we define an "efficient model" based on both performance and computational cost, such that a model is more efficient if it has better performance or lower computational cost, and vice versa.

Knowledge distillation (Hinton et al., 2015) is one way to improve model efficiency during deployment. There are several works on successful application of knowledge distillation both for pretraining tasks and for specific downstream tasks. Adhikari et al. (2020) show that knowledge distillation can be used to improve deployment efficiency of models for the downstream task of document classification in English.

In this article we investigate the effect of knowledge distillation on models for named entity recognition (NER) in Swedish.[1] The intention is to shed some light on how well knowledge distillation performs for different sequence tagging models and in the Swedish language. Our main goal is to contribute to better model efficiency within NLP. Naturally, this entails that we also focus on measuring the efficiency of each model investigated. Hopefully, this work will facilitate the development of more efficient models for both the English and the Swedish language.

2 Related work

Our work focuses on the task of named entity recognition, on model efficiency and on improving model efficiency. These topics are hardly new to the NLP arena and we will use this section to describe some of the previous work.

2.1 Named Entity Recognition

The most well-known NER task is probably the CoNLL-2003 Task created by Tjong Kim Sang

[1]The code for the project is available at https://github.com/lovhag/distilling-in-swedish.

and De Meulder (2003). It tests a model on its capacity to recognize words as either names of person (PER), location (LOC), organization (ORG), miscellaneous (MISC) or not an entity (O).

Much work has been done on NER for English, with several models trained on the data, as seen in section 2.2. However, the same cannot be said for other languages. Firstly, there is the issue of obtaining an adequate training, development and test dataset for NER. The largest Swedish dataset which can be used for NER is built on the SUC 3.0 dataset (Ejerhed et al., 1992).

NER data resources have also been developed in other North-Germanic languages and work on this is ongoing. Recently, Hvingelby et al. (2020) created a novel NER dataset for Danish. In the same article, they provide an overview of the available NER datasets for similar languages, such as Swedish and Norwegian. They also train a BERT model for their Danish NER task and obtain an f1 score of 83.76.

2.2 Named Entity Recognition models

When Devlin et al. (2019) tested their BERT model on the downstream task of NER they used the CoNLL-2003 English data and obtained an f1 score of 92.4 with their base model.

One previous state of the art model for NER before BERT, named "CCNN+WLSTM+CRF", is provided by Yang and Zhang (2018) and Ma and Hovy (2016).[2] It does not use hand-crafted features or deep contextualized word embeddings.

2.3 Model efficiency

Research that focuses on model efficiency and energy consumption is seemingly on the rise. The most noteworthy contribution within the field of NLP is that of Strubell et al. (2019). In their work, Strubell et al. claim that the NLP field would benefit from reporting training time and sensitivity to hyperparameters for developed models. Additionally, Clark et al. (2020) argue that compute efficiency should be taken in consideration together with downstream performance for representation learning methods. To this end, they report model performance as a function of train FLOPS necessary to reach that performance.

[2]According to `http://nlpprogress.com/english/named_entity_recognition.html`.

2.4 Development of more efficient models

Several methods for making models within language processing more efficient have been developed and research on this is ongoing. Seemingly, the methods so far discovered can be categorized into three different types: 1) conditional computation, 2) improving sample efficiency and 3) model compression. Conditional computation is about not using the full network when making inferences, thus reducing the number of computations needed (Shazeer et al., 2017; Fedus et al., 2021). The goal of improving sample efficiency is quite self-explanatory, and may be exemplified by the recent work by (Clark et al., 2020) in which a more effective method for training BERT is proposed. Model compression is the focus of this article and will be further explained in this section.

The objective of model compression is to compress a large model with good performance into a smaller model that still performs on par with the larger model. A "smaller model" is a model which in some way requires less computational power and/or memory. In general, this means that you still need to do some training of the larger model before you can compress it. As such, model compression is beneficial when you want to achieve energy efficiency at deployment. Apart from knowledge distillation, pruning can also be used to this end. For example, after the lottery ticket hypothesis was presented for neural networks by Frankle and Carbin (2018), Chen et al. (2020) presented corresponding work on iterative pruning for BERT models.

Knowledge distillation (KD) is another model compression technique that will be the main focus of this article. The main idea behind the technique is to distill the knowledge from a larger model, a teacher, into a smaller model, a student, by providing the student with the predictions of the teacher (Hinton et al., 2015).

KD can be implemented in different ways during training of the student model. One implementation that was used by Adhikari et al. (2020) is to train the student model to also imitate the predictions of the teacher model through an additional KD term in the loss signal. This KD loss term measures how similar the predictions of the student model $\boldsymbol{y^{(s)}}$ are to those of the teacher model $\boldsymbol{y^{(t)}}$, denoted $L_{\text{KD}}(\boldsymbol{y^{(s)}}, \boldsymbol{y^{(t)}})$. The standard loss for the task, denoted $L_{\text{task}}(\boldsymbol{y^{(s)}})$, is still included in the loss signal. Thus, the training loss during

KD can be described as below.

$$L = L_{\text{task}}(\boldsymbol{y^{(s)}}) + \lambda L_{\text{KD}}(\boldsymbol{y^{(s)}}, \boldsymbol{y^{(t)}}) \quad (1)$$

Here, λ is a tunable hyperparameter used to tune the balance between how much feedback the student model should receive from the objective of the task and how much feedback it should receive from the teacher. With a non-zero λ, the student model is partly trained to imitate the predictions of the teacher model.

KD within the scope of natural language processing can be used in either of two training situations; 1) during pre-training of a model that is intended to be transferable on several downstream tasks and 2) during fine-tuning of a model for a specific downstream task.

Previous work on KD for language models intended to be transferable is that of Sanh et al. (2019) in which a distilled version of BERT (DistilBERT) was created. DistilBERT has 40% fewer models parameters than BERT and is capable of being fine-tuned to perform well on several downstream tasks without requiring as many computations as BERT.

Previous work on KD for a specific downstream task includes that by Adhikari et al. (2020). In their work Adhikari et al. found that generally any model benefits from KD for document classification. They also found that simpler models such as logistic regression models benefit the most with respect to relative improvement in f1 score.

There is also work on trying to understand *why* models benefit from KD. The number of theoretical justifications are few, although some have been found in the recent work by Rahbar et al. (2020). On the other hand, there is more work in the area of empirical explanations. Based on empirical experiments, Yuan et al. (2020) claim that the benefits of KD mainly come from the label smoothing regularization provided by the soft targets of the teacher model, such that even a "bad" teacher can improve the performance of a student model as long as it provides soft targets. Yuan et al. also suggest that an increase in performance that is comparable to that of KD can be obtained by using "self-training" or a manually designed regularization term, without the need of a teacher model.

3 Swedish NER dataset

We use the manual NER annotations based on the SUC 3.0 dataset (Ejerhed et al., 1992) for our Swedish NER task. Before training, we reshape the data to a more suitable format for our task.

SUC 3.0	CoNLL-2003
person	PER
animal	PER
myth	PER
place	LOC
institution	ORG
product	MISC
work	MISC
event	MISC
other	MISC

Table 1: The mapping used to convert SUC 3.0 entity types to the same as those of the CoNLL-2003 data.

Resource	SUC 3.0
#tokens	1,166,593
#entity tokens	47,310
%entities	4.06

Table 2: Some general features of the SUC 3.0 NER dataset in Swedish. The number of entity tokens measures the number of tokens that make up the named entities. The percentage of entities is the number of tokens that make up entities divided by the total number of tokens in the dataset.

Firstly, the manual annotations in the SUC 3.0 data contain annotations for the entities person, animal, myth (for example "God"), place, institution, product, work, event and other. These entity categories are not found in NER datasets for other languages. In order to make better comparisons to other languages, we map the entity types in the dataset to the same types as those that can be found in the CoNLL-2003 data, as described by Table 1. We also represent the data in the IOB2 format (Tjong Kim Sang and Veenstra, 1999) and split it into 70%/10%/20% for the train/validation/test data. The splits were made with random sampling without regard to text source.

Tables 2 and 3 list some of the features of the reshaped dataset. From these tables, we can observe that the Swedish dataset is about three times larger than the CoNLL-2003 dataset, while the latter has a higher density of entities. It is worth remarking that while the English dataset was developed for NER, the SUC dataset was originally compiled for the purpose of part-of-speech tagging, with the entity annotation added later. Additionally, we can

Resource		LOC	MISC	ORG	PER	#examples
SUC 3.0	train	6,705	4,551	6,005	16,030	51,971
	dev	955	549	885	2,135	7,351
	test	1,857	1,402	1,574	4,662	14,923
	total	9,517	6,502	8,464	22,827	74,245

Table 3: The distribution of the named entities of the Swedish NER dataset.

observe from Table 3 that the Swedish dataset has quite an unbalanced entity distribution.

4 Method

The goal of this work is contribute to better model efficiency within NLP by investigating the effect of KD on different NER models in Swedish. To this end, we utilize the method for KD as presented in Section 4.1 and investigate the NER models seen in Section 4.2. The efficiency of our models is then measured as described in Section 4.3.

4.1 Application-targeted KD

The general form of the KD objective was previously introduced in Equation (1). We let the KD loss term $L_{\text{KD}}(x)$ for one batch be given by the Kullback–Leibler divergence as shown in Equation (2), similarly to what was done by Adhikari et al. (2020).

$$L_{\text{KD}}(\boldsymbol{y^{(s)}}, \boldsymbol{y^{(t)}}, \boldsymbol{w}) = \tag{2}$$
$$\sum_{n} \sum_{l:\, \boldsymbol{w_{n,l}} \neq \text{“PAD”}} \sum_{k} \frac{\boldsymbol{y^{(t)}_{n,l,k}}}{N} (\log \frac{\boldsymbol{y^{(t)}_{n,l,k}}}{\boldsymbol{y^{(s)}_{n,l,k}}})$$

$\boldsymbol{y^{(s)}}$ and $\boldsymbol{y^{(t)}}$ are the respective label probabilities of student and teacher model for each token in each batch example. The sum indices n, l, k denote the batch index, token index and label index. So $1 \leq n \leq 32$, $1 \leq l \leq 128$ and $1 \leq k \leq 9$ in the case of our work. N is the batch size and $\boldsymbol{w_{n,l}}$ denotes the token at position l in the sequence with index n in the batch.

The objective of the KL divergence is to measure the difference between the student model label probabilities and the teacher model label probabilities. It is only zero if the probabilities are identical. Neither the cross-entropy loss nor the Kullback–Leibler divergence were evaluated for padding tokens.

Another important variable for the KD is λ. This was set by studying the sizes of the two loss terms and making sure that they contributed with feedback of roughly equal magnitude, as this seemingly generated the best KD results.

Moreover, data augmentation has successfully been used for improving the performance of KD (Hinton et al., 2015). Results by Ba and Caruana (2014a) indicate that the more data, the more for the student model to learn on from the teacher model. To this end, Adhikari et al. (2020) used data augmentation during KD. However, we find it meaningful to investigate the benefits of KD before the usage of data augmentation, and will not use it in this work.

4.2 Models

All of the evaluated models and their parameters are listed in Table 4. The models were chosen with the objective of investigating the effect of KD for simpler as well as more complex models on the NER task, similarly to what was done by Adhikari et al. (2020). However, we did not include quite as simple models as those evaluated by Adhikari et al., as our sequence tagging task of NER requires a sequential model output.

Common for all models except for the CharCNNWordLSTM model is that their input is formatted by a Swedish BERT tokenizer with a vocabulary size of 50,325. As such, each embedding layer of the models expects word pieces as input and covers a vocabulary of the same size as BERT. Additionally, each training example is truncated or padded to a sequence length of 128 wordpieces and the label for an entity consisting of several word pieces is given by the label generated for the first wordpiece, similarly to the approach by Devlin et al. (2019).

The BERT model was developed with support from the Huggingface Transformers software by Wolf et al. (2020). A linear classification layer was added on top of the pre-trained Swedish base BERT model by Malmsten et al. (2020) to create a BERT model for NER in Swedish. This model was fine-tuned for 3 epochs on the Swedish NER data. A cross-entropy loss was used and all layers of the model were fine-tuned during training. This

Model name	#parameters	% in emb	infer FLOPS	% of BERT FLOPS	infer time [s]
BERT	124,107,273	31.14	2.9e10	100	0.287
Window	6,445,065	99.94	8.8e5	3e-3	0.000254
Window-B	38,670,345	99.95	5.3e6	2e-2	0.000718
LSTM-128	6,708,105	96.03	6.9e7	2e-1	0.009278
LSTM-128-B	39,571,465	97.67	2.4e8	8e-1	0.011056
LSTM-256	13,940,489	92.42	2.7e8	9e-1	0.015666
LSTM-256-B	40,755,465	94.83	5.4e8	2	0.020334
LSTM-256-2-B	42,332,425	91.30	9.4e8	3	0.035662
LSTM-256-2-drop-B	42,332,425	91.30	9.4e8	3	0.037428
CharCNNWordLSTM	27,002,212	98.75	1.0e8	4e-1	0.009930

Table 4: The number of model parameters for all models investigated. We also indicate how many percentages of the parameters are found in the word embedding layer. The infer FLOPS correspond to one forward pass of an example. The infer time is the inference time of the model for one example.

model also served as the teacher during KD training of the other models in Table 4, such that $\boldsymbol{y}^{(t)}$ in Equation (2) is given by the predictions of this model.

The Window model is a straightforward implementation of a window-based sequence labeling model with a window size of 3. This window size was found to be the best after some preliminary tuning. Furthermore, the model has an initial embedding layer with dimension $(50325, 128)$ and a final fully connected top layer which predicts for the nine available labels.

The LSTM-128 model is a straightforward implementation of an LSTM model with an initial embedding layer with dimension $(50325, 128)$, a hidden bidirectional LSTM layer with size 128 and a final fully connected top layer for the labels. The same applies for the LSTM-256 model, with the exception that the LSTM layer of this model has a size of 256 and that the embedding dimension is 256.

The LSTM-256-2 model has the same architecture as the LSTM-256 except for that it utilizes two bidirectional LSTM layers instead of one.

The LSTM-256-2-drop model has the same architecture as the LSTM-256-2 model except for that it has a word dropout probability of 0.2 and a dropout layer with a dropout probability of 0.2 on the output of the first LSTM layer. This model was chosen to investigate the effect of KD on a more regularized model.

The -B extension denotes that the same model architecture is used, but with the pre-trained word piece embedding layer of size 50325×768 from the BERT model. For these -B models the embedding layer is frozen during training. Consequently, this increases the number of parameters for the models, while it is somewhat mitigated by the fact that the embedding layer does not need to be tuned.

The CharCNNWordLSTM model was chosen with the intention of investigating the effect of KD on a state-of-the-art model for NER which does not utilize deep contextual word representations. The architecture of this model is a CharCNN+WordLSTM structure, the same as that of Yang and Zhang (2018) and Ma and Hovy (2016), with the exception that we do not include the conditional random field (CRF) layer in our CharCNNWordLSTM model. Similarly to the work by Yang and Zhang (2018) and Ma and Hovy (2016) we use pre-trained word embeddings in the model. These are given by a Word2Vec model trained on a Swedish corpus.[3] The Swedish embeddings have a word vocabulary of size 104,162 and an embedding size of 256. To make the KD from BERT feasible, the input data to the CharCNNWordLSTM model was potentially truncated to less than 128 words, since the output of it needed to be of the same shape as that of BERT which was given an input of maximum 128 word pieces. Additionally, this model is regularized with dropout and weight decay during training. This model and the LSTM-256-2-drop model are the only models with regularization mechanisms, such as dropout.

Common for all models is that none of them employ a final CRF layer. Models used for se-

[3]The corpus consists of approximately 10e9 words from a mix of corpora distributed by Språkbanken, `https://spraakbanken.gu.se/resurser`.

quence tagging usually show an improvement in performance if they have a final CRF layer which takes regard to sequential dependencies in the predictions. We chose to not use a CRF layer with the purpose of faster training and a simpler implementation of the KD.

All non-BERT models are trained and evaluated both with and without KD on a GeForce GTX TITAN X GPU. Every model was trained until it showed no further increase in f1 score. The results reported for the models in Tables 5 and 6 are the test scores for the model checkpoint with the best f1 score on the validation data. The number of epochs for the model in the table is then given by the number of train epochs required to reach this best checkpoint.

4.3 Method for measuring model efficiency

To evaluate the method of KD with respect to efficiency we measure the inference time, number of parameters as well as training and inference FLOPS required by each model investigated, as seen in Tables 4 to 6.

We use the Python package `thop` to estimate the number of FLOPS required for one forward-pass of all models in Table 4 except for BERT. These numbers are reported as "infer FLOPS" in the table. The number of FLOPS are calculated for the forward-pass of one data example with a sequence length of 128. We choose a character length of 15 for the forward-pass example in the case of the CharCNNWordLSTM model which also separates the characters of each word. We do not include the FLOPS required by the embedding layer in these calculations since we deem this number to be negligible in comparison with the FLOPS required by the other parts of the models.

To estimate the number of FLOPS required for training we then use Equation (3).[4] In the equation, n_{infer} denotes the number of FLOPS required for one forward pass and n_{examples} denotes the number of examples the model was trained on. The number of training FLOPS is reported as "FLOPS" in Tables 5 and 6.

$$n_{\text{FLOPS}} = n_{\text{infer}} \cdot 3 \cdot n_{\text{examples}} \tag{3}$$

To calculate the number of FLOPS required for one forward pass in the BERT model, which is a standard BERT-base model, we use the information given by Clark et al. (2020). The pre-train FLOPS required for BERT are then given by estimating the training parameters of the BERT training method as described by Malmsten et al. (2020) and using Equation (3) with the forward pass FLOPS previously obtained.

To calculate the inference time, denoted "infer time" in Table 4, we use the same data example as was used for calculating the number of infer FLOPS for the models. We then make the model predict for this example 100 times and estimate the average of the inference time required for each prediction iteration as the inference time of the model. These time calculations were done on a 2.3 GHz Quad-Core Intel Core i7 CPU.

5 Results and Discussion

The model scores on the Swedish NER test data are split into Tables 5 and 6. The results in the former table are of the simpler models that were not regularized, while the results in the latter are of the models that were regularized.

We split the analysis of the results with respect to our aspects of interest. Consequently, we start off with a general analysis of the model results for Swedish NER, after which we examine the effect of KD on model scores and then study the effect of KD on model efficiency.

5.1 General analysis of the Swedish NER model results

Firstly, the BERT model has the highest f1 score for the Swedish NER task. This also comes with the highest computational cost and the longest inference time, which is ten times longer than that of the second most slow model. This is not surprising, as the current trend within NLP is that better models require more resources.

Moreover, the f1 score of the BERT model is approximately two percentage units lower than that of BERT on the English NER dataset. This could be due to the difference between the datasets, different fine-tuning procedures, and/or to the different pre-training processes of the BERT models. Nonetheless, it is not entirely unexpected that the models we investigate may perform worse for the Swedish language than for the English.

[4] There is a blog post by OpenAI which explains a method for calculating model training FLOPS, see `https://openai.com/blog/ai-and-compute/`.

Model	P	R	f1	epochs	FLOPS
Window	0.667 ± 0.005	0.707 ± 0.007	0.686 ± 0.004	18 ± 5	2.4e12
KD	0.681 ± 0.006	0.705 ± 0.003	0.693 ± 0.004	18 ± 2	2.4e12
B	0.731 ± 0.000	0.721 ± 0.002	0.726 ± 0.001	24 ± 4	2.0e13
B-KD	0.726 ± 0.001	0.712 ± 0.002	0.719 ± 0.000	21 ± 2	1.8e13
LSTM-128	0.720 ± 0.004	0.717 ± 0.004	0.719 ± 0.003	60 ± 9	6.5e14
KD	0.758 ± 0.006	0.736 ± 0.005	0.747 ± 0.005	66 ± 11	7.1e14
B	0.802 ± 0.004	0.808 ± 0.005	0.805 ± 0.004	44 ± 23	1.7e15
B-KD	0.823 ± 0.003	0.822 ± 0.004	0.823 ± 0.003	60 ± 12	2.2e15
LSTM-256	0.743 ± 0.007	0.729 ± 0.008	0.735 ± 0.006	46 ± 24	1.9e15
KD	0.784 ± 0.005	0.747 ± 0.003	0.765 ± 0.003	66 ± 15	2.8e15
B	0.807 ± 0.010	0.815 ± 0.004	0.811 ± 0.006	54 ± 12	4.6e15
B-KD	0.829 ± 0.006	0.826 ± 0.002	0.828 ± 0.003	66 ± 21	5.5e15
LSTM-256-2					
B	0.830 ± 0.007	0.831 ± 0.003	0.831 ± 0.005	61 ± 21	8.9e15
B-KD	0.849 ± 0.004	0.845 ± 0.004	0.847 ± 0.004	78 ± 15	1.1e16

Table 5: The scores on the test data for all of the evaluated models that are not regularized.

Model	P	R	f1	epochs	FLOPS
BERT	0.892	0.897	0.895	3	1.4e16 (9.1e19)
LSTM-256-2-drop-B	0.844 ± 0.006	0.832 ± 0.002	0.838 ± 0.002	39 ± 9	5.7e15
KD	0.847 ± 0.004	0.833 ± 0.006	0.840 ± 0.002	25 ± 10	3.6e15
CharCNNWordLSTM	0.843 ± 0.002	0.822 ± 0.004	0.836 ± 0.008	90 ± 11	1.4e15
KD	0.842 ± 0.005	0.824 ± 0.003	0.833 ± 0.003	97 ± 2	1.5e15

Table 6: The scores on the test data for all of the evaluated models implemented with regularization. Models trained with knowledge distillation are marked with "KD". "P" denotes precision and "R" recall. Epochs, time and mean number of FLOPS required to reach best evaluation performance during training are also displayed. FLOPS values in parentheses denote number of FLOPS required during pre-training.

5.2 The effect of KD on model scores

For the one-layer LSTM models without BERT embeddings the f1 score increases with approximately 3 units when using KD training. With BERT embeddings, these models also benefit some from KD. Seemingly, the LSTM models improve primarily in precision when KD is applied.

Additionally, it appears as though the LSTM models benefit more from KD than the simpler Window model. The Window model without BERT embeddings displays an increase in precision with KD, while the same model with BERT embeddings even decreases in performance with KD. This contradicts previous results on KD by e.g. Adhikari et al. (2020), where it was found that simpler models have the most to benefit from KD. A potential reason for this could be that the model architecture was not expressive enough to benefit from KD.

Moreover, the LSTM-256-2-drop-B model performs better than its counterpart LSTM-256-2-B when no KD is applied. However, when KD is applied, the LSTM-256-2-B-KD model surpasses the LSTM-256-2-drop-B-KD model in f1 score as it seemingly benefits more from KD.

The models Window-B, LSTM-256-2-drop-B and CharCNNWordLSTM that do not clearly benefit in f1 score from KD have in common that they are either quite small or regularized. Revisiting the idea of Yuan et al. (2020), one possible reason for this is that KD provides regularization and that a model that does not need regularization consequently will not benefit in performance from KD.

5.3 The effect of KD on model efficiency

The three non-BERT models with the best f1 scores in descending order are given by the LSTM-256-2-B-KD, LSTM-256-2-drop-B-KD and the CharCNNWordLSTM models. The LSTM models are slightly better than the CharCNNWordLSTM model, although this comes with

the price of requiring approximately 4 to 10 times more FLOPS for training and an inference time that is approximately 4 times longer. The LSTM models also rely on the existence of a pre-trained BERT model, which requires approximately 9.1e19 FLOPS. While the CharCNNWordLSTM model also relies on pre-trained word embeddings, these do not require as many FLOPS.

The best non-BERT model is the LSTM-256-2-B-KD with an f1 score of 0.847. It is approximately 5 units worse than the BERT model, while it requires approximately the same number of training FLOPS (BERT pre-training not included) and only 3% of the number of inference FLOPS required by BERT. Clearly, it is more efficient at deployment, while the question remains as to whether it has a performance good enough for deployment.

The second best non-BERT model is the LSTM-256-2-drop-B-KD model. While it did not clearly benefit in f1 score from KD, it seemingly benefited with respect to the number of required training FLOPS, as the number of training FLOPS of the model decreased with approximately 40%. In every other case, the general model behavior with KD applied is that both the number of training FLOPS and the f1 score increase.

Clearly, every trained model that utilized and benefited from KD is more performance efficient than its non-distilled version when making inferences for new data, as the model improves in f1 score while the number of computations for inference is the same as before KD. However, the cost of training such a model is higher, mainly due to the need of a trained teacher model. The question is whether the gain in deployment efficiency is worth the additional effort. One way to reason about this is through basic arguments of when such an "investment" would reach a break-even point, similarly to how e.g. solar panels are judged based on how many years they would need to be used to repay the energy required to produce them. For example, if we are to develop a model that we know will be run several times during deployment, the use of KD could enable the use of a smaller model without loss of performance, thus reducing the computational cost required during deployment, weighting up for the extra cost of training it with KD. One such model that has been developed is DistilBERT (Sanh et al., 2019), which only last month was downloaded 1,544,446 times from the Huggingface model library.[5]

Apart from reasoning about model efficiency, we can also reason about when an increase in f1 score is worth the associated computational cost. Since the general trend is that we obtain better models if we allow for an increase in computational cost, the question is how much we are willing to pay for one unit of f1 score. In this case we also have to take into account that the computational cost of one f1 score unit increases with f1 score, as it is harder to increase the performance in the region of e.g. 0.9 than it is in the region of 0.6. Ethayarajh and Jurafsky (2020) propose a way to handle this by using an utility function that takes regard to performance as well as practical concerns, such as model size and inference latency. It may be appropriate to investigate the effect of KD in the eye of such an utility function.

6 Conclusion and Future work

Our work indicates that different models may differ in whether they benefit from KD. Thus, we cannot make the assumption that KD should benefit the performance of every model. Adding to the question of *why* some models seem to benefit from KD, we may also ask ourselves *in which situations* the soft targets of a teacher model may benefit a student model.

We observe three different situations for which it is worth to further investigate the effect of KD; 1) when the student model is in need of regularization, 2) when we want more data for the student model to train on and 3) when the data for training is of poor quality. The two latter situations have not been covered in this work, while they have been mentioned by other researchers (Ba and Caruana, 2014b). The former situation has already been observed by Hinton et al. (2015), and we have found additional support for it in our work. For this situation we may further investigate how KD works in combination with existing regularization techniques and whether it is a better such technique.

From our work we can also conclude that KD may provide us with more efficient models at deployment, while the cost of training these models is high due to the need of a trained teacher. This prompts us to reason about when KD is worth the effort, with regard to how we value an increase in

[5]For the uncased base version of DistilBERT.

f1 score in terms of computational costs. We also reason about situations when KD may be a good investment for models that will be used heavily during deployment. To fully measure the benefits of KD with respect to model efficiency we conclude that we need to investigate better tools for judging these trade-offs and different deployment situations.

Future work may also investigate other types of KD, such as extracting more layers than the embedding layer from the teacher model and providing teacher signals to more layers of the student model. Potentially, these KD variations could further improve the performance of a model without requiring more computational costs.

Moreover, it still remains to investigate the benefits of data augmentation for the student models during KD. The question is whether we could attain even better KD results with this approach. This could also be taken one step further to the region of completely unsupervised training on unlabeled data, merely by providing the student model with the labels generated by the teacher.

Lastly we can conclude that, unsurprisingly, KD works for the Swedish language as well. One interesting next step which may benefit the Swedish industry would be to develop a Swedish DistilBERT.

Acknowledgements

We would like to thank Tobias Norlund for his feedback during the writing process of this article. We would also like to thank the anonymous reviewers for their feedback and valuable input.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

Ashutosh Adhikari, Achyudh Ram, Raphael Tang, William L. Hamilton, and Jimmy Lin. 2020. Exploring the limits of simple learners in knowledge distillation for document classification with DocBERT. In *Proceedings of the 5th Workshop on Representation Learning for NLP*, pages 72–77, Online. Association for Computational Linguistics.

Jimmy Ba and Rich Caruana. 2014a. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, volume 27, pages 2654–2662. Curran Associates, Inc.

Jimmy Ba and Rich Caruana. 2014b. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, volume 27, pages 2654–2662. Curran Associates, Inc.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained BERT networks. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, Vancouver, Canada.

Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota.

Eva Ejerhed, Gunnel Källgren, Ola Wennstedt, and Magnus Åström. 1992. The linguistic annotation system of the Stockholm-Umeå corpus project – description and guidelines. Technical report, Department of Linguistics, Umeå University.

Kawin Ethayarajh and Dan Jurafsky. 2020. Utility is in the eye of the user: A critique of nlp leaderboard design. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4846–4853.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*.

Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Rasmus Hvingelby, Amalie Brogaard Pauli, Maria Barrett, Christina Rosted, Lasse Malm Lidegaard, and Anders Søgaard. 2020. DaNE: A named entity resource for Danish. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4597–4604, Marseille, France. European Language Resources Association.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany.

Martin Malmsten, Love Börjeson, and Chris Haffenden. 2020. Playing with words at the National Library of Sweden – making a Swedish BERT.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Arman Rahbar, Ashkan Panahi, Chiranjib Bhattacharyya, Devdatt Dubhashi, and Morteza Haghir Chehreghani. 2020. On the unreasonable effectiveness of knowledge distillation: Analysis in the kernel regime. *arXiv preprint arXiv:2003.13438*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Erik F. Tjong Kim Sang and Jorn Veenstra. 1999. Representing text chunks. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 173–179, Bergen, Norway.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Jie Yang and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of ACL 2018, System Demonstrations*, pages 74–79, Melbourne, Australia.

Li Yuan, Francis EH Tay, Guilin Li, Tao Wang, and Jiashi Feng. 2020. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3903–3911.