HOW TO DESIGN A SUSTAINABLY ACCURATE AND FAST MODEL FOR TEXT CLASSIFICATION TASKS

A PREPRINT

Moyan Mei
Course5 AI Labs
Toronto, Canada
moyan.mei@course5i.com

September 10, 2020

ABSTRACT

Empirical studies have shown that large-scale pre-trained language models such as BERT (Bidirectional Encoder Representations from Transformers) bring significant improvements. However, they are often computationally expensive in many practical scenarios, as such heavy models are not easily implemented with limited resources. Improving the efficiency of the model while maintaining its performance is then a key challenge. In this paper, I provide two "sustainable" solutions for Course5's text classification task. "Sustainable" is defined here as (1) easy to plug into any PLM (2) lower computational complexity demands (3) better performance with unlabeled data (4) efficiently uses cross-layer features.

1 Introduction

Text classification is a common problem in natural language processing, and its task is to assign a pre-defined category to a given text sequence. The key step in solving it is to learn the text representation. Mainly, there are two solutions: to use a pre-trained model or not. Previous work without pre-trained models uses various neural networks to learn textual representations, such as convolutional neural networks [9, 39, 1, 8, 25], recurrent models [13, 37, 24], and attention mechanism [35, 12].

Alternatively, extensive work has shown that it is beneficial to pre-train models on a large corpus and then utilize them to further improve downstream NLP tasks, without having to train a new model from scratch. One of the earliest pre-training models is word embedding, such as word2vec[18] and GloVe [19]. They are context-independent, so contextual word embeddings have subsequently been proposed, such as CoVe [15] and ELMo [3], which provide embeddings that capture the context of a word - its position in the sentence. More recently, pre-trained language models (PLM) have been shown to be useful for learning common language representations by exploiting large amounts of unlabeled data: e.g., OpenAI GPT [20], BERT [2], XLNet [36], etc.

Despite improvements in accuracy, these models are computationally expensive and inefficient in terms of memory consumption and latency. This shortcoming hinders their use in scenarios where speed of inference and computational cost are critical. With limited resources, it is not feasible to put them into operation. Many attempts have been made to accelerate deep model inference and reduce model size while maintaining accuracy, including quantization [5, 33, 38], weight pruning [17, 29, 4], and knowledge distillation (KD) [6, 21]. As one of the most popular methods, KD aims to reproduce the behavior of the teacher network by transferring the knowledge embedded in the larger teacher network to the smaller student network, e.g., TinyBERT [7], DistillBERT [22], PKD-BERT [26], MobileBERT [27], MKD [14], to name a few. However, it is not flexible enough to put into service for a variety of requests.

In addition, empirical study [30] found that many NLP datasets would appear to have different levels of difficulty for input samples. Heavy models (redundancy) may over-calculate simple inputs, while lightweight models tend to fail in complex samples. Based on this appeal, it is useful to design a "sustainable" model that can accommodate samples of varying complexity and gain computational efficiency with minimal loss of accuracy.

Inspired by the widely used model training strategy - Early Stopping, "sustainable" mechanisms are proposed for PLMs that adjust the number of executed layers dynamically to reduce computational steps. It enables better input adaptive inference of PLM to resolve the aforementioned limitations. Specifically, this model couples an internal classifier to each Transformer encoder layer of the PLM in order to achieve early output when certain conditions are met. Intuitively, simple input samples are easily classified through shallow layers, while difficult samples need to be passed through deeper layers for their outputs. Models trained in seven different scenarios are on average 1.06 - 12.22 times faster, and they preserve at least 97% of the performance of pre-loaded PLMs. In the following sections, I will focus on how to design and train a "sustainable" model.

2 Related Work

In summary, current research on improving the efficiency of deep neural networks can be divided into two categories: (1) static methods (2) dynamic methods. Static methods design compact models or compress heavy models. It is characterized by the fact that models will remain static while making inferences. Many lightweight neural network architectures are specifically created for resource-constrained NLP applications, such as ALBERT [10], HAT [31], DeLighT [16], LinFormer [32]etc. For model compression, Weight Quantization [5, 33, 38], Weight Pruning [17, 29, 4], Module Replacing [34], and KD [7, 22, 26, 27, 14] have proved to be effective to accelerate PLMs.

Another direction to improve the efficiency of neural networks is to implement adaptive inference on various input instances, also known as dynamic method. Early research focused on token-wise and patch-wise variations. However, these methods require more effort to train and introduce many additional parameters and inference costs. To mitigate this problem, methods such as [28, 23] compute the entropy of the predicted probability distribution as a proxy for the confidence level of the branching classifier to achieve early exit. Although branch classifiers play a key role in mitigating the behavior of the `Softmax` layer of the classification model, they are local adjustments which rely on calibrated probability of each layer for outputting a label. In addition, there is a small loss in accuracy from the given work.

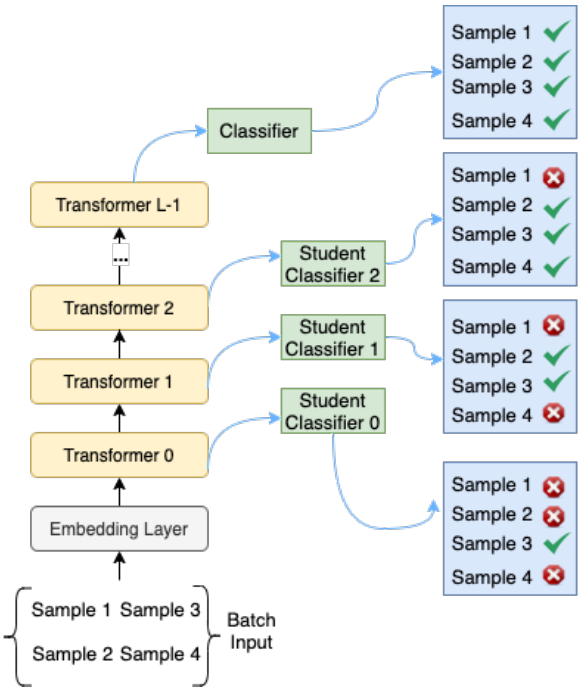

Figure 1: Inference of a sustainable model in which the number of implementation layers per sample varies according to its complexity. Taking a batch of inputs ($batch\_size = 4$) as an example, the first transformer and its student classifier outputs correct label for sample 3 easily, while other samples require more calculation through other layers to output. Cases with higher level of uncertainty is sent to the next layer for further inference, while cases with lower uncertainty are immediately used to provide the labels.

3 Methodology

In contrast to above efforts, the "sustainable" approach we consider in Course5's text classification task leverages global adjustments for early exit.

3.1 Model Architecture

As shown in Figure 2, our "sustainable" model consists of a backbone and branches. The backbone is built on top of the general PLMs (Transformers) with an additional classifier block such as BERT + Linear + `Softmax`, while the branches include student-classifiers attached to each Transformer output for early exit.

3.1.1 Backbone

The backbone consists of three parts, namely the embedding layer, the encoder containing the stacks of Transformer blocks, and the classifier block.The embedding layer and the encoder sections are identical to those of a pre-trained model, such as BERT, RoberTa, etc. Given a sequence s which consists of n tokens e.g., $s = [t_0, t_1, t_2, ..., t_n]$, its hidden representation h_i through the layer L_i is

$$h_i = L_i(h_{i-1})$$
$$h_0 = Embedding(s),$$

where the first hidden representation h_0 is the summation of word, position, and segment embeddings, and L_i is the i-th Transformer block.

After going through the last Transformer block, the final hidden representation h_{L-1} is then used as an in-domain feature vector for fine-tuning the classification task. Specifically, this process goes through a fully connected layer with a `Softmax` function that converts features from higher dimensional spaces into features from lower dimensional spaces and then into N class probabilities.

$$p = Classifier(h_{L-1})$$

3.1.2 Branches

Branches are essentially a bunch of classifiers appended after each Transformer block, which is used to mimic the work of the final classifier outputting a distribution or value as a prediction for each layer.

$$p_{s_i} = Student\_Classifier_i(h_i).$$

The student classifiers can be designed in any form to approximate the output distribution, and they do not need to be identical as each layer learns different types of features from the data. For simplicity, we consider all branch classifiers to have the same structure of the final classifier, e.g., Linear + `Softmax`.

3.2 Training

Dynamic methods such as RightTool [23] employ the so-called temperature calibration to output reliable confidence scores, so such scores can be used to determine whether it is possible to have an early exit. One obvious problem with such an approach is that those student classifiers are trained independently, thus resulting inconsistent outputs. For example, if the confidence score for layer two is high, while for layer three is low, the model will provide a label based on the output of the layer two rather than layer three. To solve the output inconsistency issue, there are two natural solutions: (1) improving the output of each layer through a priori knowledge (2) ensemble learning. Both methods take into account the effects of multiple classifiers (including the final classifier), so I address this here as global adjustments, whereas other dynamic methods mentioned above are local adjustments since they rely only on the capability of the individual branch classifiers.

It is intuitive to better train the student classifiers under the guidance of the final classifier as known as the teacher classifier. KL-Divergence is then used to minimize the discrepancy between each student classifier and the teacher classifier. Mathematically, its formula is given as

$$D_{KL}(p_s, p_t) = \sum_{i=1}^{N} p_s(i) \cdot \log \frac{p_s(i)}{p(j)}.$$

In total, there are $L-1$ student classifiers, so the sum of their KL-Divergence is used as a total loss $\updownarrow$ for training, which is formulated as

$$\mathcal{L}\left(p_{s_0}, \ldots, p_{s_{L-2}}, p_t\right) = \sum_{i=0}^{L-2} D_{KL}\left(p_{s_i}, p_t\right)$$

where p_{s_i} refers to the probability distribution of the output from the i-th student classifier. One thing to bring up is that this process only requires the final (teacher) classifier to output its soft-label p, and we are free to use an unlimited amount of unlabeled data to improve minimization of loss.

The alternative sustainable approach is simple yet efficient, and all it needs is voting (easy ensemble) over student classifiers. In this end, we only need to train each branch classifier with its cross entropy

$$\mathcal{L}_i = -\sum_{c \in \mathcal{C}} [\mathbb{1}[\mathbf{y}_i = c] \cdot \log P(\mathbf{y}_i = c \mid \mathbf{h}_i)],$$

where c is class label in a set of class labels $\mathcal{C}$. To be noted that neither method requires pre-training, as it is free to load high-quality pre-trained models.

3.2.1 Fine-tuning

We offer two kinds of fine-tuning methods,

- fine-tuning backbone and student classifiers jointly;
- fine-tuning backbone and student classifiers respectively.

The first type of fine-tuning is identical to any PLM fine-tuning in HuggingFace, which updates all layers and classifiers together. This could be more suitable for the second model since all classifiers (including the final classifier) are only concerned with the loss between their output and the true label.

On the other hand, the second kind of fine-tuning can be more beneficial for the first type model that is trained under the guidance of the final classifier because the model can be improved with unlabeled data. More specifically, the model is fine-tuned in two stages:

1. Update the weights of embedding layer, all transformer layers, and the last classifier with the loss function e.g., cross-entropy, focal loss, dice loss etc. This stage is identical to BERT fine-tuning in the original paper.
2. Freeze all fine-tuned parameters, including the final classifier for the first stage, and then update each student classifier from top to bottom with its loss e.g., KL-divergence or cross-entropy. The reason for this is to maintain the best quality of the last classifier, otherwise the Transformer encoders are no longer optimized only for the last classifier, which would generally worsen its quality.

3.3 Inference

The process of inference in both ways discussed in section 3.2 åre shown in Algorithm 1 and 2.

Algorithm 1: Given an input sequence x, its uncertainty is calculated by dividing the output probability p_{s_i} of its i-th student classifier by the normalized entropy, where N is the number of label classes. This is then used to compare with a pre-defined hyper-parameter S to determine whether a given sequence should be returned at the i-th layer or sent to the next layer. S ranges between 0 and 1, and it balances the trade-off between the speed and accuracy. S can be interpreted as the percentage of the sample that will **not** be sent to a higher layer for outputting a distribution. Both intuition and experimentation show that the larger the S, the slower but more accurate the model, and the smaller the choice S, the less accurate but faster the model. At the end, if no conditions are filled, this goes back to final classifier for prediction.

Table 1: FLOPs of each operation within the model

Operation	Sub-operation	FLOPs	Total FLOPs
Transformers	self-attention (768 -> 768)	603.0M	1809.8M
	FeedForward (768 -> 3072 -> 768)	1207.9M	
Classifier	Linear (768 -> 128)	25.1M	26.3M
	Linear (128 -> 128)	4.2M	
	Linear (128 -> N)		

To the contrary, **Algorithm** 2 allows to stop inference early at j-th layer when $cnt_j = t$, where cnt_j is the number of the times that the prediction remain unchanged, and t is the number of times it needs to be invariant. In this way, it incorporates the efforts of multiple student classifiers to output the correct label, hence ensemble learning.

Algorithm 1: Inference with entropy

Input: encoded text –> x
Output: probability –> p_{s_i} or p
for $i = 0$ *to* $L - 1$ **do**
  $p_{s_i} = f_i(x; \theta)$
  $uncertainty = \frac{\sum_{i=1}^{N} p_{s_i} \log p_{s_i}}{\log \frac{1}{N}}$
  if $uncertainty < S$ **then**
    return p_{s_i}
  end
end
return p

Algorithm 2: Inference with count

Input: encoded text –> x
Output: label –> y_i or y
for $i = 0$ *to* $L - 1$ **do**
  $y_i = \texttt{Softmax}(f_i(x; \theta))$
  if $y_i = y_{i-1}$ **then**
    $cnt_i = cnt_{i-1} + 1$
  else
    $cnt_i = cnt_{i-1}$
  end
  if cnt_i = t **then**
    return y_i
  end
end
$y = \texttt{Softmax}(p)$
return y

4 Experiment Results

4.1 Baseline and dataset

4.1.1 Baseline

In this section, we compare our two sustainable models against four baselines:

- **BERT** 12-layer BERT-base model was pre-trained on Wiki Corpus and released by Google;
- **DistillBERT** One of the most famous distillation method of BERT with 6 layers was released by HuggingFace;
- **BERT-PKD** A patient knownledge distilled method of BERT with 6 layers was released by the Microsoft;
- **BERT-of-Thesus** A compressing method of BERT with progressive module replacement was release by Microsoft Asia.

4.1.2 Dataset

To validate the effectiveness of our two sustainable models, we used six open source English datasets and one Course5 dataset. These six datasets i.e., Ag.news, Amz.F, Dbpedia, Yahoo.F, and Yelp, are sentence classification tasks and were released by [39].

Table 2: Performance comparison over seven classification datasets

Dataset/ Model	Ag.news		Amz.F		Dbpedia		Yahoo		Yelp.F		Yelp.P		Our data	
	Acc	FLOPs (speedup)	Acc	FLOPs (speedup)	Acc	FLOPs (speedup)	Acc	FLOPs (speedup)	Acc	FLOPs (speedup)	Acc	FLOPs (speedup)	Acc	FLOPs (speedup)
BERT	94.47	21785M (1.00x)	65.50	21785M (1.00x)	99.31	21785M (1.00x)	77.36	21785M (1.00x)	65.93	21785M (1.00x)	96.04	21785M (1.00x)	96.20	21785M (1.00x)
DistillBERT	94.18	10872M (2.00x)	64.05	10872M (2.00x)	99.10	10872M (2.00x)	76.73	10872M (2.00x)	64.25	10872M (2.00x)	95.31	10872M (2.00x)	94.41	10872M (2.00x)
BERT-PKD	94.31	10448M (2.08x)	64.14	10448M (2.08x)	99.16	10448M (2.08x)	76.84	10448M (2.08x)	64.44	10448M (2.08x)	95.39	10448M (2.08x)	94.52	10448M (2.08x)
BERT-of-Thesus	94.35	10642M (2.05x)	64.20	10642M (2.05x)	99.22	10642M (2.05x)	76.91	10642M (2.05x)	64.61	10642M (2.05x)	95.48	10642M (2.05x)	94.73	10642M (2.05x)
Our Model (KL) + S = 0.1	94.36	5965M (3.65x)	65.48	20869M (1.04x)	99.25	2045M (10.65x)	77.33	15984M (1.36x)	65.93	20578M (1.06x)	95.99	6628M (3.29x)	96.06	10844M (2.01x)
+ S = 0.5	93.11	2004M (10.87x)	64.44	9992M (2.18x)	99.02	1834M (11.87x)	76.83	4824M (4.52x)	64.71	9782M (2.15x)	95.32	3426M (6.36x)	95.22	5168M (4.21x)
+ S = 0.8	92.44	1783M (12.22x)	61.68	2304M (9.45x)	98.89	1833M (11.88x)	75.05	1962M (11.10x)	60.64	1945M (11.20x)	94.31	2384M (9.13x)	93.89	1864M (11.68x)
Our Model (Count) + t = 3	93.44	8284M (2.63x)	61.62	11954M (1.82x)	98.81	6450M (3.37x)	75.28	8284M (2.63x)	61.04	11954M (1.82x)	93.31	6450M (3.37x)	93.85	11954M (1.82x)
+ t = 4	94.04	10119M (2.15x)	64.18	15624M (1.43x)	98.96	8284M (2.63x)	76.44	10119M (2.15x)	63.27	15624M (1.43x)	94.24	8284M (2.63x)	95.22	15624M (1.43x)
+ t = 6	94.43	13789M (1.58x)	65.28	17458M (1.25x)	99.27	11954M (1.82x)	77.30	13789M (1.58x)	65.93	17458M (1.25x)	95.32	10119M (2.15x)	95.94	17458M (1.25x)

Our model (KL) is the first model in which the branching classifiers are trained by minimizing KL divergence under the guidance of the final classifier. The range of S is between 0 and 1. The larger the value of S, the slower the corresponding model inference, but the more accurate it is. Our model (count) addresses the outputs of the branch classifiers via a voting mechanism to determine the number of hold-ins, with t being the minimum threshold required for consistency.

4.2 FLOP analysis

Floating point operations (FLOPs) are a measure of a model's computational complexity and indicate the number of floating point operations a model performs in a process. In general, the larger the model's FLOPs, the longer the inference time. Models with low FLOPs are more efficient and more suitable for industrial use, given the same accuracy. Here, we use BERT as the backbone for illustration of FLOPs in Table 1. It is not hard to find out that the FLOPs of the classifier is way much lighter than the transformer layers. Although additional parameters (e.g. branch classifiers) are added, it achieves acceleration by reducing the large number of computation in Transformers. Note that the added parameters are less than 0.005% of the parameters in BERT and less than 1% of all FLOPs.

4.3 Training Setting

We add classifier blocks (e.g., Linear + `Softmax`) as internal classifiers after each transformation layer of the pre-trained BERT. We perform Bayesian search at learning rates between 2e-4 and 8e-5, and then we apply Adam or AdaX [11] for optimization. Experimentally, we run 10 epochs, saving one checkpoint every 1500 steps. Finally, we select the best performing model on the development set. All experiments are performed on a server with two Nvidia GTX 2080 11GB GPUs.

4.4 Overall Comparison

We present our two sustainable models versus other baseline models from the following two perspectives, speed and accuracy. The results of comparisons are shown in Table 2, where the *speedup* is obtained by using BERT as the benchmark. It can be observed that our two sustainable models based on KL-divergence and count outperform all comparison methods in improving the inference efficiency of PLMs, while retaining more than 98% of the performance of the original model. Overall, the first mode can speed up 1.06 to 12.22 times faster than BERT, and the second mode can accelerate 1.25 to 3.37 times faster. The reason why the FLOPs vary from different tasks with both our models is because of the difficulty of the dataset, which is mentioned in the aforementioned section 1. The easier the input sample, the earlier the model output its prediction. The results show that all models have relatively low accuracy scores on the datasets Amz.F and Yelp.F, which means that all models have some uncertainty to output the correct labels. As a consequence, our two models will be affected, requiring more computational complexity, i.e., more FLOPs. On the other hand, the open-source datasets with relatively high performance will not demand much FLOPs, so they can have more efficient inference. Our data is imbalanced in Course5, where the majority class is 6 to 7 times larger than the minimum two minority classes, and there are some hard examples between classes. To some extent, this can also be

seen as a sample of inputs with varying degrees of difficulty. Thus, in our data, the two models will also have relatively high FLOPs. Notably, the performance (accuracy) of the first model is very close to the original BERT. As mentioned in section 3.2, this model can be further improved with a large amount of unlabeled data.

5 Discussion

In this paper, these two models considered are solutions in Course5's products, which yield better accuracy-speed trade-off than existing methods. They are sustainable because

1. it is easy to plug in any PLM, which saves us the time of training the model from scratch.
2. it enhances the efficiency of inference by performing adaptive inference, which requires low computational complexity.
3. its performance can be further improved by the use of large amounts of unlabeled data;
4. it uses cross-layers features to output a consistent distribution given an input sample.

In addition, both models have a very practical feature in industrial scenarios, i.e., its inference can be tuned by its hyper-parameter S or t. The two models show promising results on six open source datasets and on Course5's internal dataset. The empirical results show that the two models are 1.06-12.22 times and 1.25-3.37 times faster than the baseline BERT model, respectively. In fact, we also employ other PLMs as backbones in our Course5 products, such as MiniLM, Electra, and ERNIE-2.0.

6 Future Work

For future work, we would like to explore these methods on other NLP tasks, such as named entity recognition, question answering, machine translation, etc. Currently, we are using counts to perform consistent output (simple ensembles), and it would be interesting to design other techniques to benefit more from ensemble learning. In addition, probability calibration can be considered as one of the extensions to further improve the model.

References

[1] Alexis Conneau, Holger Schwenk, Loïc Barrault and Yann Lecun. Very Deep Convolutional Networks for Text Classification. *arXiv:1606.01781*, 2016

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805*, 2018.

[3] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee and Luke Zettlemoyer. Deep contextualized word representations. *arXiv:1802.05365*, 2018.

[4] Angela Fan, Edouard Grave and Armand Joulin. Reducing Transformer Depth on Demand with Structured Dropout. *arXiv:1909.11556*, 2019.

[5] Yunchao Gong, Liu Liu, Ming Yang and Lubomir Bourdev. Compressing Deep Convolutional Networks using Vector Quantization. *arXiv:1412.6115*, 2014.

[6] Geoffrey Hinton, Oriol Vinyals and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531*, 2015.

[7] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang and Qun Liu. TinyBERT: Distilling BERT for Natural Language Understanding. *arXiv:1909.10351*, 2019.

[8] Rie Johnson and Tong Zhang. Deep pyramid convolutional neural networks for text categorization. *ACL*, 2017.

[9] Nal Kalchbrenner, Edward Grefenstette and Phil Blunsom. A Convolutional Neural Network for Modelling Sentences. *arXiv:1404.2188*, 2014.

[10] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *arXiv:1909.11942*, 2019.

[11] Wenjie Li, Zhaoyang Zhang, Xinjiang Wang and Ping Luo. AdaX: Adaptive Gradient Descent with Exponential Long Term Memory. *arXiv:2004.09740*, 2020

[12] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou and Yoshua Bengio. A Structured Self-attentive Sentence Embedding. *arXiv:1703.03130*, 2017.

[13] Pengfei Liu, Xipeng Qiu and Xuanjing Huang. Recurrent Neural Network for Text Classification with Multi-Task Learning. *arXiv:1605.05101*, 2016.

[14] Linqing Liu, Huan Wang, Jimmy Lin, Richard Socher and Caiming Xiong. MKD: a Multi-Task Knowledge Distillation Approach for Pretrained Language Models. *arXiv:1911.03588*, 2019.

[15] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *NIPS*, 2017.

[16] Sachin Mehta, Marjan Ghazvininejad, Srinivasan Iyer, Luke Zettlemoyer and Hannaneh Hajishirzi. DeLighT: Very Deep and Light-weight Transformer. *arXiv:2008.00623*, 2020.

[17] Paul Michel, Omer Levy and Graham Neubig. Are Sixteen Heads Really Better than One? *arXiv:1905.10650*, 2019.

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *NIPS*, 2013.

[19] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. *EMNLP* 2014.

[20] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *Technical report, OpenAI*, 2018.

[21] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta and Yoshua Bengio. FitNets: Hints for Thin Deep Nets. *arXiv:1412.6550*, 2014.

[22] Victor Sanh, Lysandre Debut, Julien Chaumond and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108*, 2019.

[23] Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge and Noah A. Smith. The Right Tool for the Job: Matching Model and Instance Complexities. *arXiv:2004.07453*, 2020.

[24] Minjoon Seo, Sewon Min, Ali Farhadi and Hannaneh Hajishirzi. Neural Speed Reading via Skim-RNN. *arXiv:1711.02085*, 2017.

[25] Dinghan Shen, Yizhe Zhang, Ricardo Henao, Qinliang Su, and Lawrence Carin. Deconvolutional latent-variable model for text sequence matching. *AAAI Conference on Artificial Intelligence*, 2018.

[26] Siqi Sun, Yu Cheng, Zhe Gan and Jingjing Liu. Patient Knowledge Distillation for BERT Model Compression. *arXiv:1908.09355*, 2019.

[27] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang and Denny Zhou. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. *arXiv:2004.02984*, 2020.

[28] Surat Teerapittayanon, Bradley McDanel and H. T. Kung. BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks. *arXiv:1709.01686*, 2017.

[29] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich and Ivan Titov. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. *arXiv:1905.09418*, 2019.

[30] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *EMNLP*, 2018.

[31] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, Song Han. HAT: Hardware-Aware Transformers for Efficient Natural Language Processing. *ACL*, 2020

[32] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang and Hao Ma. Linformer: Self-Attention with Linear Complexity. *arXiv:2006.04768*, 2020.

[33] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. *CVPR*, 2016.

[34] Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei and Ming Zhou. BERT-of-Theseus: Compressing BERT by Progressive Module Replacing. *arXiv:2002.02925*, 2020.

[35] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. *ACL* 2016.

[36] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv:1906.08237*, 2019.

[37] Dani Yogatama, Chris Dyer, Wang Ling and Phil Blunsom. Generative and Discriminative Text Classification with Recurrent Neural Networks. *arXiv:1703.01898*, 2017.

[38] Ofir Zafrir, Guy Boudoukh, Peter Izsak and Moshe Wasserblat. Q8BERT: Quantized 8Bit BERT. *arXiv:1910.06188*, 2019.

[39] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *NIPS*, 2015.