Compress Reordered Transformer via Meta-Learning

Anonymous ACL submission

Abstract

Beyond the success in recent deep learning community, block-stacked Transformer-based models are blamed for the large scale of parameters, which limits their practical utility when deployed on devices with computational resource limitations. Though many compression methods have been proposed to cope with this inconvenience, they encounter poor generalization ability on unseen data. In this paper, we propose a novel compressor for Transformer architecture in the light of Meta-Learning (*Meta-Compressor*), which prunes unhelpful model weights and meanwhile largely ensures the generalization ability at the same time. *Meta-Compressor* is updated by measuring the performance loss before and after compression on out-of-bag data to enhance its performance and generalization ability. We conduct empirical experiments on machine translation and sentence classification tasks. In machine translation, we successfully prune the standard Transformer into 56% of its original size, and maintain 92% of the performance, while in sentence classification, our pruned Transformer demonstrates even stronger generalization on the challenging dataset.

1 Introduction

Transformer (Vaswani et al., 2017) has achieved great success since its proposal. Attention mechanism, the core idea of Transformer, has breathed new life into natural language processing (NLP) area, laying a solid basis for a vast majority of pre-trained language models such as BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019), ELECTRA (Clark et al., 2020) and DeBERTa (He et al., 2020). However, these models usually have an extremely large number of parameters, causing unaffordable computing and memory requirements, which makes these models difficult to be loaded offline or on portable devices. At the same time, when huge models are dealing with a small amount of data, they are susceptible to over-fitting, which greatly deteriorates their generalization ability. Therefore, compressing model parameters while maintaining performances is of vital importance for downstream applications.

A mass of model compression methods have been proposed to reduce model size and accelerate model inference, including knowledge distillation (Hinton et al., 2015; Sanh et al., 2019; Jiao et al., 2019), parameter sharing and factorizing embedding layers (Lan et al., 2019), word-vector elimination (Goyal et al., 2020) and attention head pruning (Michel et al., 2019; Voita et al., 2019). However, these methods usually directly prune the parameters with low weights without considering information from out-of-bag data, causing their poor ability to handle with out-of-distribution data, i.e., quite unsatisfactory generalization ability.

To remedy this, we seek help from a general-purpose, universal machine learning technique: Meta-Learning (Thrun and Pratt, 1998). Meta-learning, which means 'learning to learn', was born with the expectation that the model can acquire the learning ability of human, so that the model can quickly learn new tasks on the basis of acquiring existing knowledge. Meta-learning tasks are also divided into training and testing stages like common machine learning tasks. The model is trained on training stage and evaluated on testing stage. In this way, after certain training, the model can perform well on tasks for which it was not trained. Our goal is to make the compressor perform well on out-of-distribution data, so we can use meta-learning to construct our compressor. Based on meta-learning described in (Finn et al., 2017), we proposed *Meta-Compressor* to build a concise model from the original large Transformer. In meta-learning, the training and testing tasks are somewhat different. To achieve this condition, we separates the dataset at the time of compression and the dataset used to update the compres-

sor. In experiment, we train Transformer model on training dataset and update compressor according to Transformer model's performance on development dataset after compression.

While updating *Meta-Compressor*, we design two parts of loss function: 1) the performance loss of Transformer on development dataset; 2) the l_1 loss of compression parameters in *Meta-Compressor*. The former aims at better performance, while the latter aims at inducing sparsity in compression parameters in *Meta-Compressor* and identifying which parameters in the Transformer need to be pruned. In order to facilitate the compression of the dimensions of feed-forward sub-layers together, we use another technique described in (Press et al., 2019): sandwich. We rearrange all sub-layers of Transformer, putting the feed-forward sub-layers together. In this way, the dimension of hidden states spread between sub-layers does not need to change to fit in the dimension of parameter tensors of attention sub-layers.

Meta-Compressor can compress the amount of feed-forward sub-layer parameters to a very low level, while still maintaining a good performance. For the hidden states dimension of Transformer, we can even compress it from 512 to single digits while maintain an acceptable performance. In addition to good performance after compression, *Meta-Compressor* can also improve model robustness. In the experiment, *Meta-Compressor* can even achieve better results with a smaller model on sentence classification task, whose training dataset and development dataset is adversarial. We use development dataset in the experiment, but actually we can use any other datasets while evaluating performance and updating *Meta-Compressor*.

2 Related work

2.1 Meta-learning

In the long history, many people have explored in meta-learning (Thrun and Pratt, 1998). Here we discuss some related work. In (Ravi and Larochelle, 2016), an LSTM-based optimizer was designed to learn the update rule for few shot learning. In (Finn et al., 2017), Model-Agnostic Meta-Learning (MAML) was proposed to deal with any model trained with gradient descent by learning an optimized initialization ready to fast adapt to new tasks within few steps. Moreover, many other scenarios also draw on the idea of learning at two-stages and a variety of methods and models using meta-learning have been proposed in recent years. (Lee et al., 2019) learns to perturb latent features for generalization, (Lv et al., 2020; Ke et al., 2020) use MAML-inspired pre-training to find a shared representation of both language modeling and downstream tasks, and (Kang et al., 2020) designs a mask generator for masked language model.

2.2 Model Compression Method

With the increasing size of the model, model compression has become more and more popular and has become an important research topic. There are a variety of model compression methods. Knowledge distillation (Hinton et al., 2015) transfers the knowledge embedded in a large teacher network to a small student network and train the student network to reproduce the behavior of the teacher network. ALBERT (Lan et al., 2019) uses parameter sharing and factorizing embedding layers to reduce model size and accelerate model inference. Quantization (Shen et al., 2020; Zafrir et al., 2019) also injects new inspiration into the field of model compression. For Transformer models, attention head pruning (Michel et al., 2019; Voita et al., 2019) also acts as an effective compression method.

3 Method

In this section, we will specifically describe how the compression algorithm and meta-learning are combined in *Meta-Compressor*.

3.1 Pre-processing Before Compression

Transformer (Vaswani et al., 2017) consists of two parts: encoder and decoder, and each part is composed of several identical layers. Each layer consists of two sub-layers: an attention sub-layer and a feed-forward sub-layer. The input and output dimension of all these sub-layers is h. Each feed-forward sub-layers contains two fully-connected layers: F_1 and F_2. The dimension of F_1 is $h \times f$ and the dimension of F_2 is $f \times h$.

The compressor we propose tries to compress the h dimension of the original Transformer. To make preparations for compression, we consider the method in (Press et al., 2019): rearrange all sub-layers in Transformer. It is proposed that putting the self-attention sub-layer more in the front and putting the feed-forward sub-layer more in the back can achieve better performance. So

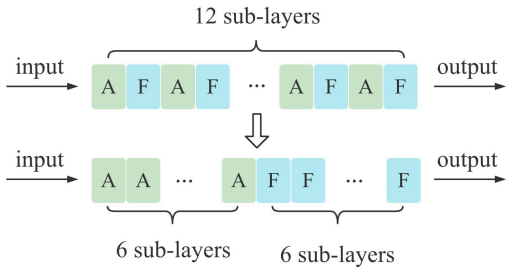

Figure 1: Structure adjustment of encoder or decoder: 'A' refers to attention sub-layer (or attention sub-layer+cross-attention sub-layer) and 'F' refers to feed-forward sub-layer

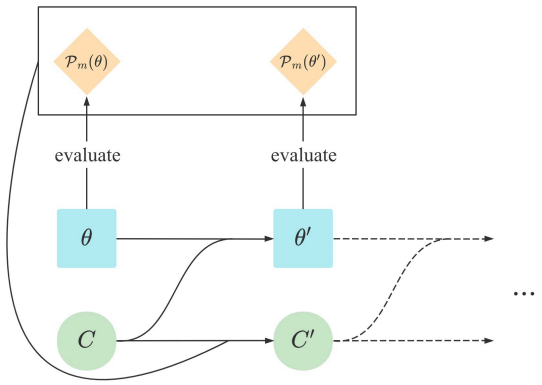

Figure 2: Method: 1) use compressor C and model θ to get the compressed model θ'; 2) evaluate the two models on development dataset and get the result $\mathcal{P}_m(\theta)$ and $\mathcal{P}_m(\theta')$; 3) use the result to update compressor and get C'; 4) repeat the above steps.

we put all of the self-attention sub-layers to the front and put all of the feed-forward sub-layers to the end. For the decoder part containing the cross-attention sub-layer, we group the attention layer and the cross-attention layer as described in (Press et al., 2019), and put all of them to the front. This change of architecture is shown in Figure 1. In our preliminary experiment, we have reached the conclusion that this will not hurt the performance. At the same time, putting all feed-forward sub-layers together can help us focus on compressing the h dimension of parameter tensors in them, because doing so can avoid the dimension change across different type of sub-layers.

3.2 Compression

Our *Meta-Compressor* aims at compressing the h dimension of Transformer as mentioned above. Using meta-learning method (Finn et al., 2017), to update our compressor, we get feedback on compression effect from out-of-bag data. Dataset is divided into two parts: training dataset and development dataset. The original Transformer model is trained on training dataset and evaluated on development dataset. One core part of the loss function to update the compressor is performance loss on development dataset. We take the difference between the results obtained by the model before and after compression as part of it. We hope that the compression process can minimize the performance loss, and even achieve better results. Specifically, suppose that for the model θ, its performance is recorded as $\mathcal{P}_m(\theta)$. The model before compression is recorded as θ, and the model after compression is θ'. Then the first part of loss function is shown in Equation 1.

$$\mathcal{L}(\theta', \theta) = \mathcal{P}_m(\theta) - \mathcal{P}_m(\theta') \tag{1}$$

Considering how to compress the model, we refer to the method in (Khetan and Karnin, 2020). Suppose there are n feed-forward sub-layers in the encoder. In the compressor, we use pruning parameters $\alpha_{hi} \in \mathbf{R}^h (i \in [n])$ to indicate which rows/columns of the parameter tensors in the feed-forward sub-layers is being pruned. α_{hi} is initialized to $\mathbf{1}$. In the training, we diagonalize α_{hi} and multiply it by the parameter tensors in the ith feed-forward sub-layer. We use these new tensors in the training. The l_1 penalty of the parameters in the compressor used for compression acts as the other part of loss function, hoping to induce sparsity in compression parameters in *Meta-Compressor*, so that we can better choose the parameters in Transformer that need to be kept. The second part of loss function, which strengthen the capability of *Meta-Compressor* to selecting compression parameters, is shown in Equation 2.

$$\mathcal{L}_1(\theta_c) = \sum_i \|\alpha_i\| \tag{2}$$

Combining the above two parts, the total loss function is shown in Equation 3, in which γ is a hyper-parameter.

$$\mathcal{L}_c(\theta', \theta, \theta_c) = \mathcal{L}(\theta', \theta) + \gamma \mathcal{L}_1(\theta_c) \tag{3}$$

And we update the compressor θ_c by Equation 4, in which β is the learning rate of compressor.

$$\theta_c \leftarrow \theta_c - \beta \nabla_{\theta_c} \mathcal{L}_c(\theta', \theta, \theta_c) \tag{4}$$

In this way, we update the compressor to compress more data and to achieve better results. The process is shown in Figure 2.

After updating the compressor, we use the compressor to prune the model. We set the smallest part of α_{hi} to zero, and then remove the corresponding rows/columns in the Transformer model. In this way, we can get a smaller model.

In order to make the hidden states spread between layers adapt to the dimension after compression, we additionally add two fully-connected layers before and after all feed-forward sub-layers respectively. The former changes the dimension h of the output of attention layers to the smaller dimension h', and the latter change the dimension h' back to the dimension h in order to fit into the evaluation program.

4 Experiment

In this section, we evaluate the effectiveness of *Meta-Compressor* with different tasks and compression settings. Our implementations are based on PyTorch using *transformers* toolkit[1]. Both the baseline model and the compressed model are trained from scratch and trained for the same number of epochs using the same dataset.

4.1 Datasets

Our experiment datasets and tasks include: 1) machine translation task: bilingual evaluation understudy (BLEU) on dataset IWSLT14 En-De; 2) sentence classification task: Paraphrase Adversaries from Word Scrambling based on Quora Question Pairs (PAWS-QQP) (Zhang et al., 2019).

4.2 Setup

Parameters In all of the tasks, we compress for the hidden size h in Transformer. We use the following compression parameters:

1. $h_{reduction}$: an integer that means how many of the original h dimension we delete at each compression.

2. γ: the hyper-parameter in Equation 3. It is set to 0.01 in all of the tasks.

3. compression gap g: an integer that means we train g epochs and compress once.

For example, if $h_{reduction}$ is 5, compression gap g is 2, after training for 20 epochs, we perform $20/2 = 10$ compressions. Then the original parameter tensors whose dimension is $[h, f]$ in feed-forward sub-layers will be reduced to $[h - 5 \times 10, f]$.

[1] https://github.com/huggingface/transformers/

Machine translation task In this task, the encoder and decoder of the Transformer we use have 6 attention sub-layers, 6 feed-forward sub-layers and 8 attention heads. The hidden size h we compress for is 512. In the experiment, the maximum sequence length is 128, batch size is 128, and learning rate of Transformer and compressor is 10^{-4}.

Sentence classification task This task acts as an adversarial task, in which data in training dataset and development dataset is adversarial, making it difficult for the model to predict the results. In this task, the encoder of Transformer we use has 12 attention sub-layers, 12 feed-forward sub-layers and 12 attention heads. The hidden size h we compress for is 768. In the experiment, the maximum sequence length is 128, batch size is 32, learning rate of Transformer is 5×10^{-5} and learning rate of compressor is 10^{-4}. The number of training epochs is 10. The compression gap in this task is set to 2.

Compressor As mentioned above, we put the feed-forward sub-layers all to the end of the encoder and decoder. And we initialize the α_{hi} in the compressor to $\mathbf{1}$. For machine translation task, we use two compressors to compress encoder and decoder respectively. We use training dataset to train the model and use development dataset to update the compressor.

4.3 Results

Machine translation task First, we test the standard Transformer and the Transformer with only layer rearrangement on dataset IWSLT14 En-De. The results are shown in Table 1. The standard model is slightly smaller than model with layer rearrangement because the latter includes two fully-connected layers before and after all feed-forward sub-layers as mentioned above. Their performance is almost the same. So we can safely rearrange the layers to facilitate our compression and in the subsequent machine translation experiment, we use the model with layer rearrangement as the baseline.

Next comes our experimental part. Using the settings in Section 4.2, we conducted several experiments with several compression rates on datasetn IWSLT14 En-De, the results are shown in Table 2. *Meta-Compressor* achieves great performance. When the model size is reduced to 56% of its original size, its BLEU is only reduced 8%.

Model	model size (M)	BLEU
standard	51.5	31.42
layer rearrangement	52.6	31.63

Table 1: Performance of standard Transformer and Transformer with only layer rearrangement on dataset IWSLT14 En-De.

$h_{reduction}$	model size (M)	BLEU
0	52.6	31.63
1	51.6	30.91
5	48.0	30.55
10	43.3	30.03
15	38.7	29.60
20	34.1	29.31
23	31.4	29.34
24	30.4	29.59
25	29.5	29.11
26	28.6	28.79
27	27.7	27.73
28	26.7	26.52

Table 2: Performance of model with different $h_{reduction}$ on dataset IWSLT14 En-De. The model whose $h_{reduction}$ is 0 means the Transformer with only sub-layers rearrangement. The compression gap is set to 2.

$h_{reduction}$	Model size (M)	Accuracy (%)
baseline	110.7	71.79
30	99.4	71.79
50	91.8	71.79
80	80.6	71.79
100	73.0	78.73
120	65.5	78.29
140	58.0	78.14
150	54.2	75.33

Table 3: Performance of model with different size on PAWS-QQP

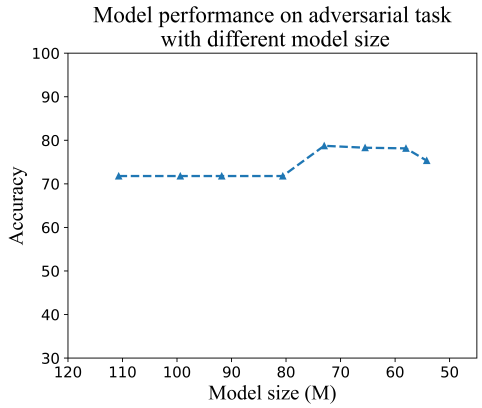

Figure 3: Model performance on PAWS-QQP with different model size

Sentence classification task In the experiment on dataset PAWS-QQP, without compression, we find that the accuracy of the model remains the same, at a relatively low level. After testing for various $h_{reduction}$, we find that *Meta-Compressor* can handle this problem well. The results are shown in Table 3. In the experiment, when the reduction ratio is low, the accuracy remains at a low level. But when we reduce hidden size to a small number, the accuracy increases. As is shown in Table 3, when we set $h_{reduction}$ to 100, and the model size is compressed from 110.7M to 73.0M, the accuracy is increased from 71.79% to 78.73%. And as is shown in Figure 3, when we gradually increase $h_{reduction}$, the accuracy remains higher than the baseline, while the model size can even be halved. This is because the original model is too large and there are too many neurons, resulting in a large part of the mechanical copying of other neurons. Although this can perform well on training, it does not work on out-of-bag samples. Thus, by pruning, we increase the generalization ability of the model. Meanwhile, in the experiments, we use the development dataset in meta-leading, which contains adversarial data. The information of the adversarial data is obtained by the model, and thus robustness of the model is also improved. It can be seen from the above experimental results and analysis that *Meta-Compressor* well boosts the generalization ability and robustness of the model, so after compression, the accuracy of the model goes up.

5 Ablation studies

In this section, we conduct ablation studies to investigate the influence of: a) different reduction ratio which intuitively affects model size; b) different compression gap which affects the training time and the adaptation time of the compressed model; c) out-of-bag data used in updating compressor.

5.1 Effects of Reduction Ratio

$h_{reduction}$ is the hyper-parameter that affects model size. In this part, we test on the dataset IWSLT14 En-De and the experiment setup is the

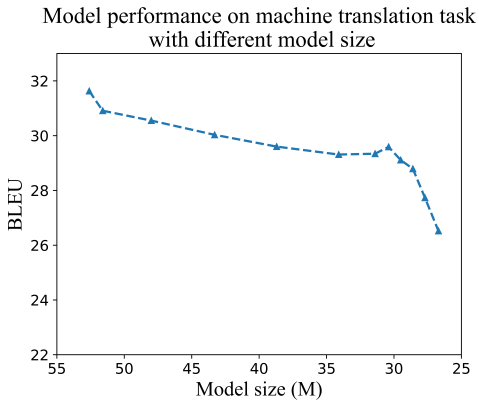

Figure 4: Model performance on dataset IWSLT14 En-De with different $h_{reduction}$

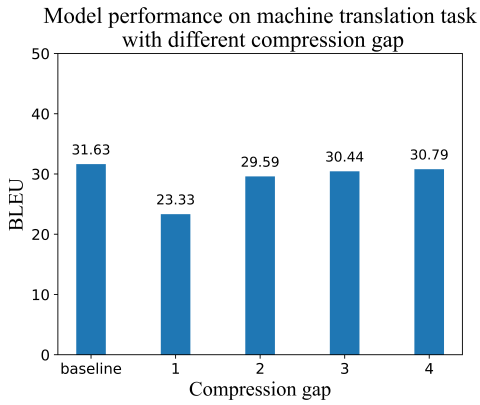

Figure 5: Model performance on dataset IWSLT14 En-De with different compression gap

same as described in Section 4.2. In this section, by changing $h_{reduction}$, we use our method to compress the model to different degrees and test their performance on machine translation task using dataset IWSLT14 En-De. In the experiment, we set training epoch to 18 and the compression gap to 2. Table 2 and Figure 4 demonstrate the results of different $h_{reduction}$. *Meta-Compressor* performs well on different $h_{reduction}$ and we highlight that when $h_{reduction}$ is further increased within a certain range and the model size is further reduced, the performance loss of the model is very small. When we focus on the case where $h_{reduction}$ is large, we can come to the conclusion that BLEU remains at an acceptable level, which demonstrates the excellent compression capability of *Meta-Compressor*. But compared to the performance when $h_{reduction} \leq 25$, the model performance drops sharply. This is because the hidden size h is reduced to too small a number, affecting the transmission of information during layers.

5.2 Effects of Compression Gap

In the experiment, we find that the compression gap influence the performance greatly. So in this section, we test different compression gaps. In this part, we test on machine translation task using dataset IWSLT14 En-De and the experiment setup is the same as described in Section 4.2. In the experiment, we set $h_{reduction}$ to 24 and number of compression to 18, so the model size before and after compression is 52.6M and 30.4M. We test four compression gaps: 1, 2, 3 and 4. The results are shown in Table 4 and Figure 5. With the increase of compression gap, the performance of

compression gap	BLEU
1	23.33
2	29.59
3	30.44
4	30.79
baseline	31.63

Table 4: Performance of model with different $h_{reduction}$ on dataset IWSLT14 En-De. The model whose $h_{reduction}$ is 0 means the Transformer with only sub-layers rearrangement. The $h_{reduction}$ is set to 24.

the model is better. But when compression gap is relatively large, as compression gap increases, the performance of the model will not increase much, but our training cost and duration will be much higher. It is worth noting that when we set compression gap to 1, the performance loss is huge, which means adjustment after compression is still necessary.

5.3 Effects of Out-of-bag Dataset

While doing compression using *Meta-Compressor*, we take advantage of out-of-bag data to update compressor, which brought vital information in the model. In this part, we test the importance of out-of-bag data for performance improvement. The dataset we use is PAWS-QQP and the experiment setup is the same as described in Section 4.2.

We set $h_{reduction}$ to 100, 120, 140, 150, which show a performance improvement in the previous experiment. In experiment with out-of-bag data, we still update the compressor with the development dataset, and in the experiment without out-of-bag data, we evaluate the pre- and post-

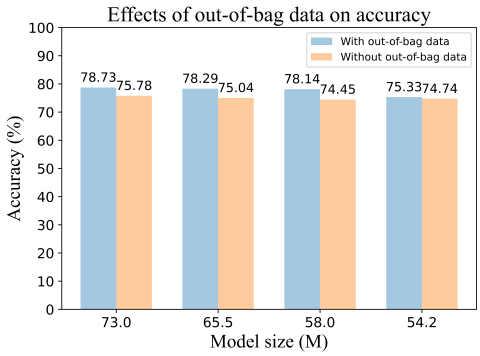

Figure 6: Model performance on PAWS-QQP with and without out-of-bag data

compression performance with the same amount of data in the training dataset as the development dataset and update the compressor accordingly. The accuracy of models compressed using these two methods is shown in Table 5.

As can be seen from the results, our compression itself can improve the generalization ability and robustness of the model, reduce the risk of overfitting, and allow the model to have a higher accuracy when encountering adversarial samples. At the same time, our compressor makes good use of out-of-bag data, so that the accuracy of the model is greatly improved compared to when there is no out-of-bag data.

Faced with a variety of out-of-bag data, our compressor can handle it well, and use the information of out-of-bag data to compress the model in the direction of better performance. So compared to other compression methods, our compressor not only allows the model to maintain good performance after compression, but also utilizes out-of-bag data, which other compression methods cannot achieve. As a result, our compressor improves the generalization performance and robustness of the model.

6 Conclusion

In this paper, we proposed *Meta-Compressor*, a novel meta-learning based compression method to compress reordered Transformer, which learns from out-of-bag data and updates through measuring the performance loss before and after model compression. The experiment on machine translation task proves its excellent compression ability, while the experiment on adversarial sentence classification task demonstrates its improvement to model generalization and robustness.

As independent of any specific model or task nature, our *Meta-Compressor* is actually universal for a variety of compression algorithms. In the future, the specific compression method can take advantage of other compression methods, and combine with meta-learning to achieve a better compression effect. And we can also try to eliminate the number of layers. By merging layers, *Meta-Compressor* may achieve even smaller model and better performance.

References

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. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR.

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

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

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

Minki Kang, Moonsu Han, and Sung Ju Hwang. 2020. Neural mask generator: Learning to generate adaptive word maskings for language model adaptation. *arXiv preprint arXiv:2010.02705*.

Zhen Ke, Liang Shi, Songtao Sun, Erli Meng, Bin Wang, and Xipeng Qiu. 2020. Pre-training with meta learning for chinese word segmentation. *arXiv preprint arXiv:2010.12272*.

Ashish Khetan and Zohar Karnin. 2020. schubert: Optimizing elements of bert. *arXiv preprint arXiv:2005.06628*.

$h_{reduction}$	With out-of-bag data?	Model size (M)	Accuracy (%)
baseline	-	110.7	71.79
100	√	73.0	78.73
	×		75.78
120	√	65.5	78.29
	×		75.04
140	√	58.0	78.14
	×		74.45
150	√	54.2	75.33
	×		74.74

Table 5: Performance of models with or without out-of-bag data

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Hae Beom Lee, Taewook Nam, Eunho Yang, and Sung Ju Hwang. 2019. Meta dropout: Learning to perturb latent features for generalization. In *International Conference on Learning Representations*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Shangwen Lv, Yuechen Wang, Daya Guo, Duyu Tang, Nan Duan, Fuqing Zhu, Ming Gong, Linjun Shou, Ryan Ma, Daxin Jiang, et al. 2020. Pre-training text representations as meta learning. *arXiv preprint arXiv:2004.05568*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*.

Ofir Press, Noah A Smith, and Omer Levy. 2019. Improving transformer models by reordering their sublayers. *arXiv preprint arXiv:1911.03864*.

Sachin Ravi and Hugo Larochelle. 2016. Optimization as a model for few-shot learning.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Sebastian Thrun and Lorien Pratt. 1998. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*.

Yuan Zhang, Jason Baldridge, and Luheng He. 2019. Paws: Paraphrase adversaries from word scrambling. *arXiv preprint arXiv:1904.01130*.