EBERT: Efficient BERT Inference with Dynamic Structured Pruning

Zejian Liu[1,2], **Fanrong Li**[1,2], **Gang Li**[1] **and Jian Cheng**[1,2]
[1]National Laboratory of Pattern Recognition, Institute of Automation, CAS
[2]School of Future Technology, University of Chinese Academy of Sciences
{liuzejian2018, lifanrong2017}@ia.ac.cn, gangli0426@gmail.com,
jcheng@nlpr.ia.ac.cn

Abstract

Pruning has been demonstrated as an effective way of reducing computational complexity for deep networks, especially CNNs for computer vision tasks. In this paper, we investigate the opportunity to accelerate the inference of large-scale pre-trained language model via pruning. We propose EBERT, a dynamic structured pruning algorithm for efficient BERT inference. Unlike previous methods that randomly prune the model weights for static inference, EBERT dynamically determines and prunes the unimportant heads in multi-head self-attention layers and the unimportant structured computations in feed-forward network for each input sample at run-time. Experimental results show that our proposed EBERT outperforms other state-of-the-art methods on different tasks.

1 Introduction

In the last few years, transformer-based (Vaswani et al., 2017) large-scale pre-trained language models, such as BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and GPT-3 (Brown et al., 2020), have achieved state-of-the-art results on many NLP tasks, including language understanding, question answering, and reading comprehension. Most recently, researchers also successfully applied transformer-based models to computer vision tasks, achieving comparable or superior performance compared to traditional convolutional networks. For example, Carion et al. (2020) propose detection transformer (DETR) for object detection, Dosovitskiy et al. (2021) design a transformer-based model, namely Vision Transformer (ViT), for image classification. However, due to the notable computational complexity and memory footprint, it is difficult for these models to deploy on hardware platforms under moderate computing and resource budget. Therefore, how to reduce model complexity to enable efficient inference for large-scale pre-trained language models is a critical issue.

Pruning is a commonly used technique for network compression, which has been widely explored to reduce computation and storage requirements of convolutional neural networks for computer vision tasks (Han et al., 2015, 2016; Li et al., 2016). However, *can transformer-based models benefit from pruning?* Michel et al. (2019) observe that a large percentage of attention heads can be removed with negligible performance drop, which indicates that the importance of different heads in same layer is different. Sanh et al. (2020) propose a simple, deterministic first-order weight pruning method which can prune lots of parameters with minimal accuracy loss. Although these methods are able to reduce the memory footprint, they cannot achieve real performance gain on general-purpose hardware, such as GPGPU, due to the unstructured sparsity after pruning.

Adaptive inference strategy is also proposed to accelerate the inference of BERT. It is based on two observations: 1) the input samples usually have different levels of difficulty. For a given model, it may over-calculate the simple samples while fail in complex samples (Liu et al., 2020); 2) similar to convolutional neural networks, the lower and higher layers of transformer extract different information, and features provided by the intermediate layers may be enough for some samples (Xin et al., 2020). FastBERT (Liu et al., 2020) and DeeBERT (Xin et al., 2020) are two state-of-the-art adaptive inference models for compressing BERT. Both of them insert extra classification layers between each layer of the network. During inference, each input sample only goes through part of model when the outputs of extra classifiers meet predefined criteria like entropy and uncertainty. Because the number of executed layers is reduced, real speedup can be achieved. However, skipping all the computations

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 4814–4823

of the remaining layers may be harmful to the accuracy.

In this paper, we propose EBERT, a hardware-friendly, simple yet effective algorithm that incorporates structured pruning with adaptive inference for efficient BERT inference. Specifically, EBERT inserts predictors for self-attention sub-layer and feed-forward sub-layer in each transformer block, as illustrated in Figure 1. During inference, the predictors dynamically determine which heads of self-attention layers and channels of feed-forward network can be pruned according to current input. Once a head or a channel is pruned, the corresponding computations and memory cost can be completely avoided. Compared with static pruning methods that permanently prune some parameters, it can avoid prune important parameters for current input samples which will cause large performance drop. To the best of our knowledge, it's the first time to apply dynamically structured pruning to BERT. Experimental results on different benchmark demonstrate that the proposed EBERT can achieve better trade-off between computation reduction and accuracy.

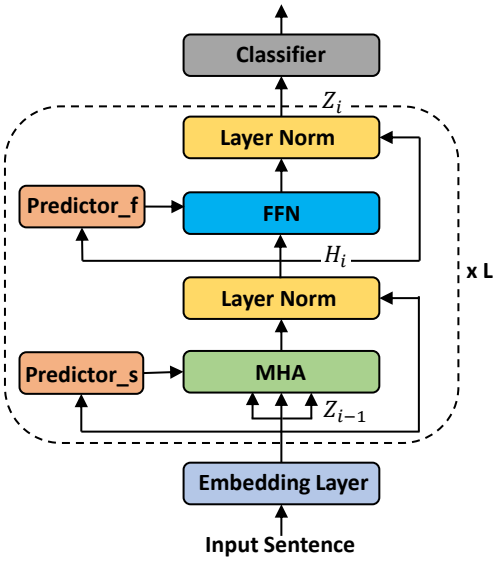


Figure 1: The overall architecture of EBERT. For each input sentence, the predictor dynamically determines which heads or channels can be pruned.

2 Related Work

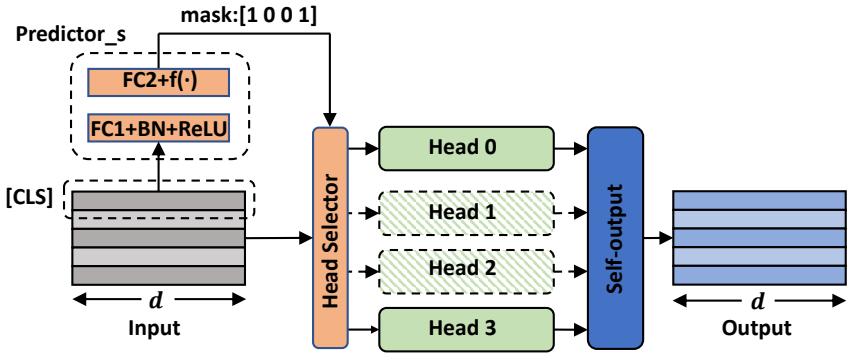
Adaptive inference. As different input samples usually have different levels of difficulty, using fixed-size model to process all samples may be non-optimal in terms of computational efficiency. Therefore, the main goal of adaptive inference is to adaptively skip part of the computations according to each input sample to reduce complexity. FastBERT (Liu et al., 2020) adds student classifiers to the output of each transformer block and use self-distillation strategy to improve performance. The model architecture of DeeBERT (Xin et al., 2020) is similar to FastBERT, but it use entropy of output to decide whether to exit at early stages. PABEE (Zhou et al., 2020) proposes a novel early-exit criterion that dynamically stops forward computing when the output of internal classifiers keep unchanged for a pre-defined number of steps.

Pruning. Pruning is an intuitively simple yet effective technique for model compression, which removes unimportant computations based on certain criterion. Michel et al. (2019) observe that a large percentage of attention heads can be removed with negligible performance loss and propose a greedy pruning algorithm. Compressing BERT (Gordon et al., 2020) explores the effect of unstructured weight pruning with different levels of pruning and different training stages. McCarley et al. (2019) investigate the relationship between structured pruning and task-specific distillation. SNIP (Lin et al., 2020) proposes a structured pruning method to penalize an entire residual module in Transformer model toward an identity mapping.

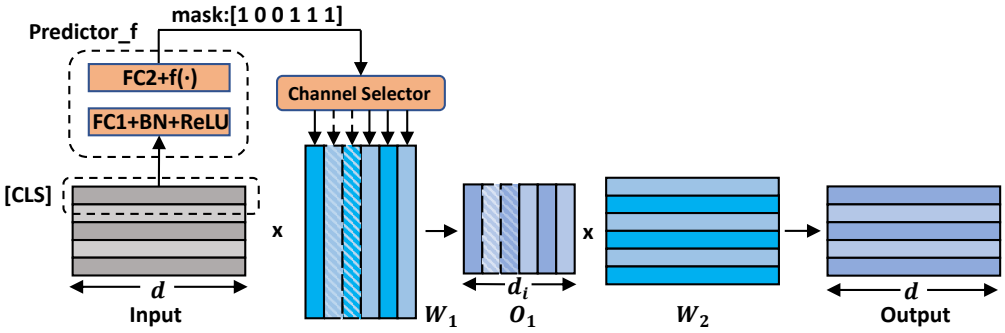
Distillation. Knowledge Distillation (Hinton et al., 2015) is an effective technique to get light models from heavy models without sacrificing too much performance. DistilBERT (Sanh et al., 2019) leverages knowledge distillation at pre-training phase to get a lighter pre-trained model, then directly fine-tunes on downstream tasks. BERT-PKD (Sun et al., 2019) proposes an incremental knowledge extraction process. Apart from learning from the final output of teacher model, student model also patiently learns from intermediate layers. TinyBERT (Jiao et al., 2020) performs distillation at both the pre-training and task-specific fine-tuning phase. Data augmentation is also used to improve the accuracy of student model.

3 Methods

In this section, we will first introduce the architecture of EBERT. As shown in Figure 1, it can be divided into BERT branch and predictor branch. Then we will describe the training and inference in details.



(a) Predictor_s in MHA.



(b) Predictor_f in FFN.

Figure 2: The details of predictors in MHA and FFN layer. Here we assume that $n = 4$, $h = 4$ and $d_i = 6$. Shadow area means that computations can be skipped.

3.1 BERT Branch

The architecture of BERT consists of three parts: the embedding layer, multi-layer bidirectional Transformer encoders and the task-specific classification layer. Given an input sentence $S = [s_0, s_1, ..., s_n]$ with length n, where s_0 is usually a special classification token $[CLS]$, the embedding layer will transform it to a sequence of vector representations:

$$E = Embedding(S), E \in R^{n \times d} \quad (1)$$

The Transformer encoder contains two sub-layers: multi-head self-attention (MHA) layer and position-wise fully connected feed-forward network (FFN),

$$\begin{aligned} H_i &= LN(MHA(Z_{i-1}) + Z_{i-1}) \\ Z_i &= LN(FFN(H_i) + H_i) \end{aligned} \quad (2)$$

where $i = 1, 2, ..., L$ and $Z_0 = E$. LN is the Layer Normalization operation.

The final component of BERT is a task-specific classification layer. It accepts the representation to $[CLS]$ token as input to generate final results, as:

$$O = Classifier(Z_L[0, :]) \quad (3)$$

3.2 Predictor Branch

In order to prune unimportant heads and channels for individual input sentence, we add predictors for MHA and FFN in each layer, respectively. The predictor consists of two feed-forward layer, one batch normalization layer and a ReLU activation layer, as depicted in Figure 2. The output t of the second feed-forward layer will be transformed to a 0-1 mask by a function $f(\cdot)$:

$$\begin{aligned} t &= FC2(ReLU(BN(FC1(x)))) \\ m &= f(t), m \in \{0, 1\} \end{aligned} \quad (4)$$

where $x = Z[0, :]$. It means that the input of predictor is only $[CLS]$ representation. This choice is based on two reasons. 1) *Overhead.* Although using the whole representation of input sentence may improve the performance of predictors, the amount of computations increases linearly with the sentence length n. When n is large, the computational overhead of predictors can not be ignored. 2) *Representation ability.* Because the final hidden state to $[CLS]$ token in the last transformer block is used in task-specific classifier to generate classification results, we assume that $[CLS]$ repre-

sentation encodes most of the useful information of the sentence. Note that the representation to $[CLS]$ token in the first MHA is independent with the input sentence, so we use average pooling of MHA as input.

Intuitively, t represents the probability of heads or channels being selected. In order to train the model end-to-end with back propagation, Gumbel-Softmax trick (Jang et al., 2017; Maddison et al., 2016) is adopted in our model. Given class probabilities $\pi_1, \pi_2, ...\pi_n$, discrete samples z can be drawn as:

$$z = one\_hot(\arg\max_i[g_i + log\pi_i]) \tag{5}$$

where g_i is a sample drawn from a Gumbel distribution. Gumbel-Softmax trick replaces $\arg\max$ operation with a softmax function, which is a continuous differentiable approximation to $\arg\max$:

$$y_i = \frac{exp((log(\pi_i) + g_i)/\tau}{\sum_{j=1}^{k} exp((log(\pi_j) + g_j)/\tau)} \tag{6}$$

As the value of mask m is binary (0 for prune and 1 for preserve), we can simplify the Gumbel-Softmax formulation (Verelst and Tuytelaars, 2020). For the output $t[i] \in (-\infty, \infty)$, we can convert it to probabilities π_1 and π_2 by using a sigmoid function σ:

$$\begin{aligned} \pi_1 &= \sigma(t[i]) \\ \pi_2 &= 1 - \sigma(t[i]) \end{aligned} \tag{7}$$

Substituting (7) into (6), we can get:

$$\begin{aligned} y_1 &= \sigma(\frac{t[i] + g_1 - g_2}{\tau}) \\ y_2 &= 1 - y_1 \end{aligned} \tag{8}$$

As $y_1 < y_2$ means the head or channel will be pruned, the final formulation is:

$$f(t[i]) = \begin{cases} 1, & if \quad y_1 > 0.5 \\ 0, & otherwise \end{cases} \tag{9}$$

3.3 Training

The entire training process can be divided into three stages: fine-tune the BERT branch, joint train both branches, and re-train the BERT branch.

Fine-tuning. In the first stage, only BERT branch is fine-tuned on downstream tasks with loss $\mathcal{L}_{task}$. The training strategy is the same as BERT in (Devlin et al., 2019).

Joint Training. In this stage, we jointly train the pre-trained BERT branch and randomly initialized predictor branch to make the average ratio of remaining Floating-point operations (FLOPs) reach a target value $C_t \in [0, 1]$. In order to achieve this goal, we add a loss to minimize the difference between real computational cost of the whole network and C_t:

$$\mathcal{L}_s = \left(\frac{F_c}{F_o} - C_t\right)^2 \tag{10}$$

Where F_o is the FLOPs of original network, and F_c is the average FLOPs of current model in a mini-batch.

In addition to the FLOPs constraint, we also add extra loss function to control the sparsity of each MHA and FFN, as in (11). The purpose is to avoid high sparsity of some layers that is harmful to the accuracy of the model.

$$\begin{aligned} \mathcal{L}_M &= \frac{1}{L}\sum_{l=0}^{L-1}\left(\frac{F_c^{lM}}{F_o^{lM}} - C_t\right)^2 \\ \mathcal{L}_F &= \frac{1}{L}\sum_{l=0}^{L-1}\left(\frac{F_c^{lF}}{F_o^{lF}} - C_t\right)^2 \end{aligned} \tag{11}$$

where F_c^{lM} and F_o^{lM} refer to the FLOPs of l-th MHA in current model and original model. The definition of F_c^{lF} and F_o^{lF} is similar. The final loss to be optimized is then given by

$$\mathcal{L} = \mathcal{L}_{task} + \lambda_1\mathcal{L}_s + \lambda_2(\mathcal{L}_M + \mathcal{L}_F) \tag{12}$$

where λ_1 and λ_2 control the magnitude of task and sparsity loss, respectively.

Re-training. As different input samples usually activate different parts of heads, the total update of a particular head is less than that of regular training process. As a result, the heads are probably not trained sufficiently. So do the channels in FFNs. Therefore, in this stage, we freeze the parameters of predictors and only re-train the BERT branch.

3.4 Inference

The computation flow during inference is shown in Figure 2. Given an input sequence, the predictor generates a mask by using the representation to $[CLS]$ token. For MHA, heads with mask '0' will not be executed. For FFN, as matrix-matrix multiplication can be transformed to multiple matrix-vector multiplications, we only need to complete part of computations where vector's mask is not zero.

BERT-base	L=12, h=12, d=768, $d_i = 3072$
Predictor_s	$768 \rightarrow 64 \rightarrow 12$
Predictor_f	$768 \rightarrow 64 \rightarrow 3072$

Table 1: The detailed setting of BERT and Predictors. RoBERTa is with the same setting.

Note that the exponential operation in (8) is typically expensive on hardware. Fortunately, this formulation can be simplified during inference by removing Gumbel noise. $f(\cdot)$ now can be rewritten as:

$$f(t[i]) = \begin{cases} 1, & if \quad t[i] > 0 \\ 0, & otherwise \end{cases} \tag{13}$$

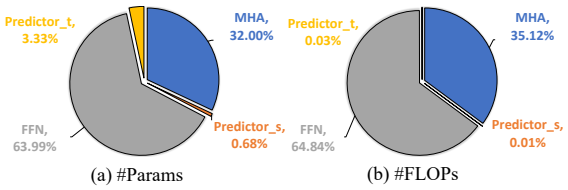
4 Experiments

4.1 Setup

Datasets and Metrics. To verify the effectiveness of EBERT, we conduct experiments on four classification tasks from GLUE benchmark (Wang et al., 2018): Multi-Genre Natural Language Inference Matched/Mismatched (MNLI-m/mm), Quora Question Pairs (QQP), Question Natural Language Inference (QNLI) and Stanford Sentiment Treebank (SST-2). We exclude other tasks as the results have large variance due to the number of training examples is very small (less than 9k). MNLI-m/mm, SST-2 and QNLI use accuracy as metric, while the average of F1 and accuracy is used for QQP.

Furthermore, we also conduct experiments on SQuAD1.1 (Rajpurkar et al., 2016) and SQuAD2.0 (Rajpurkar et al., 2018), both of which are large-scale reading comprehension datasets. SQuAD1.1 consists of more than 100k questions, and the answer to each question is a segment of text from the corresponding reading passage. SQuAD2.0 is more difficult as it contains over 50k unanswerable questions. We mainly report Exact Match (EM) and F1 scores.

Implementation details. We apply the proposed methods to both BERT-base and RoBERTa-base, and implement them with the HuggingFace Transformers Library (Wolf et al., 2020). The detailed setting of BERT and predictors is shown in Table 1. Figure 3 shows the ratio of FLOPs and parameters of each operation in one encoder. We can find that the extra cost of the predictors is very small. All experiments are completed on a single Nvidia GeForce RTX2080Ti GPU.



(a) #Params (b) #FLOPs

Figure 3: The ratio of FLOPs and parameters of each operation in one encoder.

For the GLUE benchmark, we set batch size to 32, learning rate to 3e-5, training epochs to 3 while other hyperparameters are kept unchanged from the library for all downstream tasks at backbone fine-tune stage. During joint training, we use $\lambda_1 = 4, \lambda_2 = 20$ for BERT while $\lambda_1 = 2, \lambda_2 = 10$ for RoBERTa. The learning rate for predictors' parameters is 0.02 and 0.01, respectively. The hyperparameters in the third stage is the same as the first stage.

For SQuAD1.1 and SQuAD2.0, the batch size is 12, learning rate is 3e-5 and training epoch is 2. Other settings are consistent with those for BERT on GLUE benchmark.

Baseline. In order to evaluate the effectiveness of EBERT, we implement a Top-k version of BERT that $f(\cdot)$ is as (14). We keep the sparsity of each layer the same, so the value of k can be decided by C_t. What's more, for a certain k, the sparsity is a fixed value, so no extra loss need to be added. The training objective is just $\mathcal{L}_{task}$. The training methods is the same as EBERT with Gumbel-Softmax.

$$f(t[i], k) = \begin{cases} 1, & if \quad t[i] \in topk(t) \\ 0, & otherwise. \end{cases} \tag{14}$$

For convenience, in the following sections we will use the subscript t to represent Top-k version and use subscript g for Gumbel-Softmax version.

4.2 Results on the GLUE benchmark

The main results of our proposed method on the development set of GLUE benchmark are shown in Figure 4. For BERT-base, the results of Gumbel-Softmax is always better than Top-k with the same or even smaller ratio of remaining FLOPs on four tasks. For example, when remaining 50% FLOPs, EBERT_g only drops 0.6% on QQP task, while EBERT_t drops 1.8%. On the MNLI task, EBERT_g's accuracy with 77% remaining FLOPs is higher than the accuracy of EBERT_t with 81% remaining FLOPs.

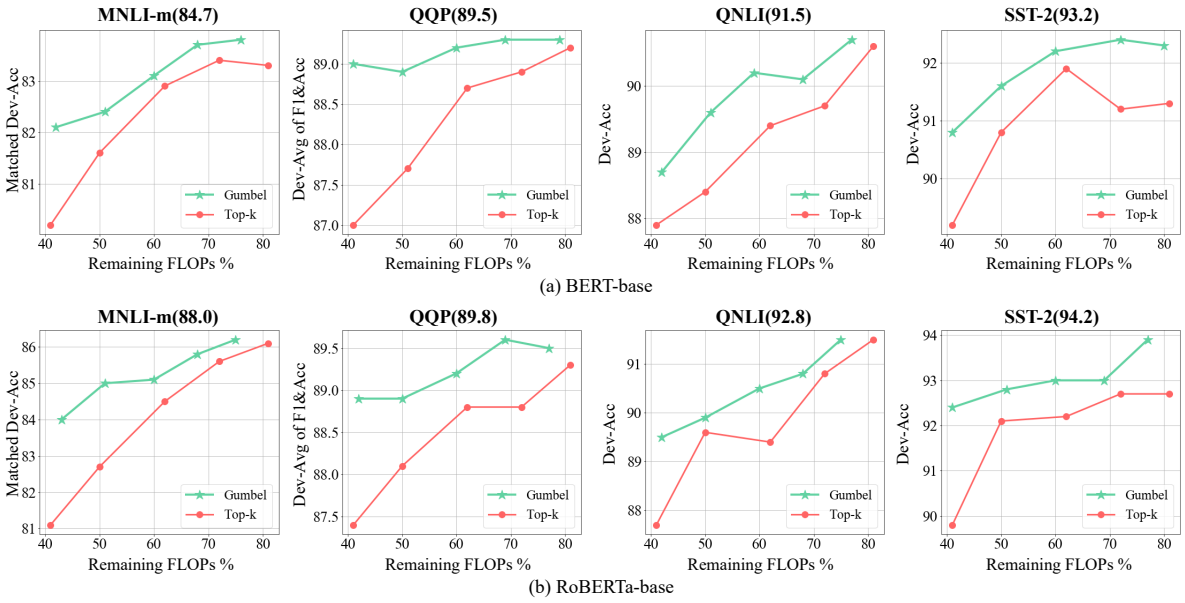


Figure 4: Results on the development set of GLUE benchmark.

Figure 4(b) shows the performance of ERoBERTa, and we can find the similar result, *e.g.* with 50% remaining FLOPs, the performance of ERoBERTa$_g$ is 2.3% higher than ERoBERTa$_t$ on the MNLI task. This proves the generality of our proposed method to different model.

4.3 Results on the SQuAD benchmark

To further demonstrate the generality of our method, we conduct experiments on the SQuAD v1.1 and v2.0 benchmark, which are reading comprehension task that the model need to predict the answer text span in the text for a given question. The results are shown in Figure 5. Similar to the observation in Figure 4, our approach achieves consistent improvement on each ratio of remaining FLOPs compared with the Top-k version. For instance, with 50% remaining FLOPs, EBERT$_g$ improves the EM and F1 score by 2.8% and 2.4% on SQuAD v1.1, respectively. On SQuAD v2.0, the improvement of EM and F1 score is 3.3% and 3.4%.

4.4 Comparison with Other Methods

We compare our proposed EBERT with other state-of-the-art compression methods. For distillation methods, we compare with DistilBERT (Sanh et al., 2019), BERT-PKD(Sun et al., 2019) and BERT-of-Theseus (Xu et al., 2020). For pruning, we compare with SNIP (Lin et al., 2020). We also compare with other two dynamic methods: DeeBERT (Xin et al., 2020) and PABEE (Zhou et al., 2020). We do not compare with FastBERT (Liu et al., 2020) as they don't report results on the GLUE and SQuAD benchmark.

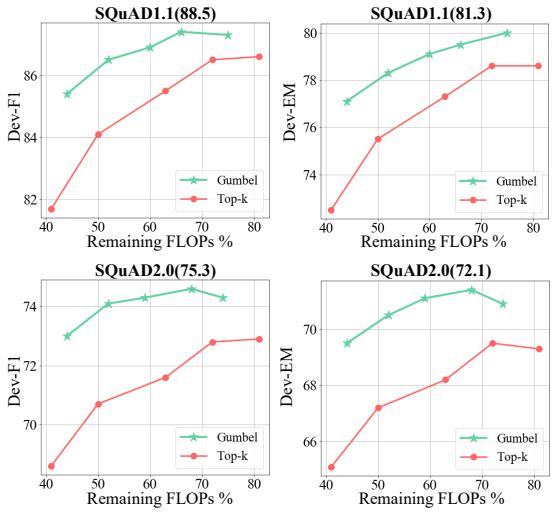


Figure 5: Results on the development set of SQuAD benchmark.

Note that other works don't report the FLOPs. However, as all of these methods try to reduce computational cost by reducing the number of layers dynamically or statically, it is reasonable to get FLOPs from speedup ratio or compression ratio under the assumption that the FLOPs is proportional to the execution time for a specific layer. For example, as the DistilBERT-6L only has half number of layers of BERT-base, we assume the ratio of remaining FLOPs is 50%.

	MNLI-m		SST-2		QQP		QNLI	
	Acc.	RF%.	Acc.	RF%.	F1/Acc.	RF%.	Acc.	RF%.
dev set								
BERT-base	84.7	100	93.2	100	87.9/91.1	100	91.5	100
DistilBERT-6L	82.2	50	91.3	50	-/88.5	50	89.2	50
BERT-PKD	81.3	50	91.3	50	-	-	-	-
SNIP	-	-	91.8	50	-/88.9	50	89.5	50
DeeBERT	80.7	63	90.0	63	-	-	-	-
PABEE	**83.6**	62	92.0	62	-	-	-	-
EBERT_g	82.4	51	91.6	50	87.2/90.6	50	89.6	51
	83.1	60	**92.2**	60	**87.5/90.8**	59	**90.2**	59
test set								
BERT-base	84.7	100	93.7	100	71.5/89.4	100	90.8	100
BERT-PKD	81.5	50	92.0	50	70.7/88.9	50	89.0	50
BERT-of-Theseus	82.4	50	92.2	50	**71.6/89.3**	50	**89.6**	50
DeeBERT	80.0	63	91.5	53	69.4/-	51	87.3	56
EBERT_g	82.4	50	92.8	50	70.1/88.8	50	89.2	50
	83.3	60	**93.4**	60	70.0/88.8	59	**89.6**	59

Table 2: Comparison with other compressed methods on the development and test set of MNLI, SST-2, QQP and QNLI. RF means the ratio of remaining FLOPs.

Table 2 lists the results on both development set and test set. The results on test set are provided by the GLUE evaluation server. Compared with other methods, our approach retains competitive performance with less FLOPs. For instance, our approach achieves the accuracy of 92.2% on SST-2 with 60% remaining FLOPs. On the test set of MNLI task, the accuracy of our method is 83.3% with only 60% remaining FLOPs, while DeeBERT' accuracy is 80.0% with 63% remaining FLOPs.

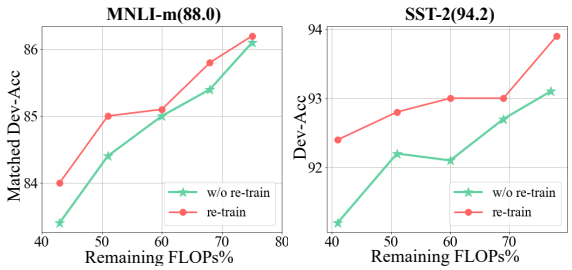


Figure 6: The effectiveness of re-training stage for RoBERTa on MNLI and SST-2.

4.5 Further Analysis

4.5.1 Impact of Re-training

The training process of EBERT contains three stages: fine-tuning, joint training and re-training. The purpose of re-training is to make each head and channel sufficiently trained. To evaluate the efficacy of this stage, we conduct experiments with RoBERTa on two tasks. Results are shown in Figure 6, we can see that the performance improvement is obvious. With 50% remaining FLOPs, the performance of the model is improved from 84.4% to 85.0% on MNLI and 92.2% to 92.8% on SST-2, respectively. The average performance improvement on MNLI and SST-2 is 0.4% and 0.8%, respectively. Comparing these two results, we find that the improvement is more obvious on small datasets. The reason for this phenomenon is that the parameters of the model are updated more frequently on large datasets, which makes the training of the model more sufficient at the joint training stage. As a result, re-training can be skipped for large datasets to make trade-offs.

4.5.2 Mask Distribution

Like in (Chen et al., 2019), we investigate the distribution of the learned masks. Although EBERT can dynamically generate mask for each head and channel for different samples, some masks may be constant of all time, which means that these masks are input-independent. Figure 8 is the layer-wise visualization of mask distribution in MHA and FFN on SST-2 task for masks that are 1) always one (on), 2) always zero (off), and 3) input-dependent. We can see that a large subset of the masks are input-dependent for both heads and channels, which indicates that our model learns to predict the im-

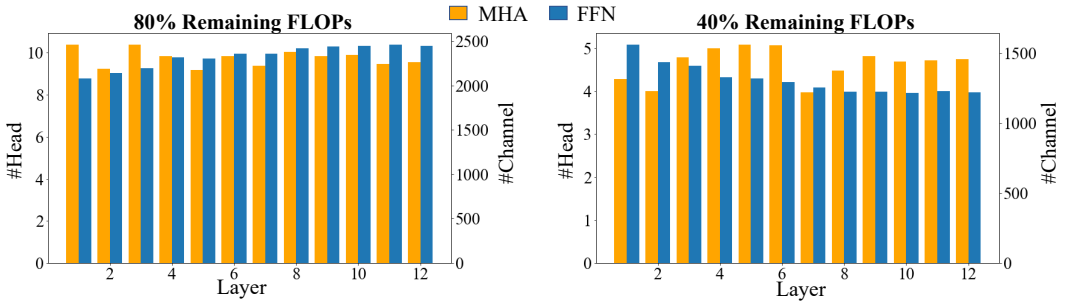


Figure 7: Average number of non-pruned heads of MHA and non-pruned channels of FFN by layer for RoBERTa-base with different remaining FLOPs on the SST-2 task.

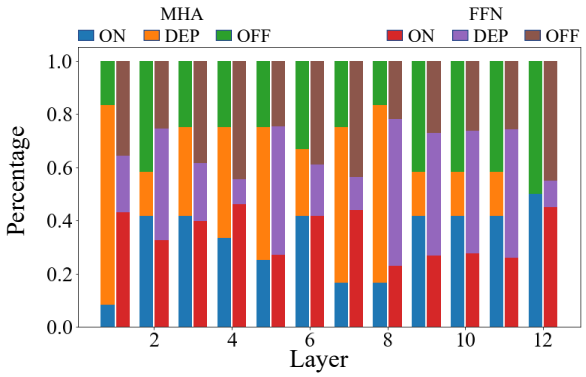


Figure 8: The distribution of masks in MHA and FFN for RoBERTa-base with 50% remaining FLOPs on SST-2 tasks. DEP refers to input-dependent.

portance of heads and channels for different input samples. For head, the proportion of masks that are input-dependent is higher in the shallow layers. For channel, the 2nd, 5th, 8th and 11th layer have higher proportion of input-dependent masks than other layers.

4.5.3 Layer Distribution

In Section 3.3, we add two extra loss $\mathcal{L}_M$ and $\mathcal{L}_F$ to prevent some layers from being too sparse. We conduct experiments on SST-2 task with RoBERTa to verify the effectiveness of these constraints. Figure 7 shows the average number of non-pruned heads of MHA and non-pruned channels of FFN with different ratio of remaining FLOPs. We can see that the number in each layer is quite close, which indicates the average amount of calculations is similar. More importantly, this value is near the target C_t. For example, when remaining 80% FLOPs, the number of non-pruned heads is around 9, which is exactly 80% of the number of heads in one MHA. Similarly, the number of non-pruned heads are around 4 and 5 when remaining 40% FLOPs. This phenomenon proves that $\mathcal{L}_M$ and $\mathcal{L}_F$ do limit the sparsity of each layer.

5 Conclusion and Future Works

In this paper, we propose a novel pruning method for efficient BERT inference, which is called EBERT. With the help of predictor branch, EBERT can dynamically prune unimportant heads in MHA and unimportant channels in FFN for each input sample at run-time. Compared with other compression methods, experiments on GLUE and SQuAD benchmarks demonstrate that EBERT can achieve better accuracy-efficiency trade-off.

As we talk about in Section 4.1, the performance of our method on small dataset has large variance. Similar observations also have been mentioned in other works (e.g. SNIP). In order to improve the generality of our method, it would be interesting to find out the exact reason and find the corresponding solution.

Acknowledgment

This work was supported in part by National Natural Science Foundation of China (No.61972396), National Key Research and Development Program of China (No. 2020AAA0103402), the Strategic Priority Research Program of Chinese Academy of Sciences (No. XDA27040300).

References

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. *arXiv preprint arXiv:2005.14165*.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *ECCV*, pages 213–229, Cham. Springer International Publishing.

Zhourong Chen, Yang Li, Samy Bengio, and Si Si. 2019. You look twice: Gaternet for dynamic filter selection in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

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. Association for Computational Linguistics.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Mitchell A. Gordon, Kevin Duh, and Nicholas Andrews. 2020. Compressing bert: Studying the effects of weight pruning on transfer learning. *arXiv preprint arXiv:2002.08307*.

Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations (ICLR)*.

Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, volume 28, pages 1135–1143. Curran Associates, Inc.

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

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710.

Zi Lin, Jeremiah Liu, Zi Yang, Nan Hua, and Dan Roth. 2020. Pruning redundant mappings in transformer models via spectral-normalized identity prior. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 719–730. Association for Computational Linguistics.

Weijie Liu, Peng Zhou, Zhiruo Wang, Zhe Zhao, Haotang Deng, and Qi Ju. 2020. FastBERT: a self-distilling BERT with adaptive inference time. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6035–6044, Online. Association for Computational Linguistics.

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*.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables.

J.S. McCarley, Rishav Chakravarti, and Avirup Sil. 2019. Structed pruning a bert-based question answering model. *arXiv preprint arXiv:1910.06360*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, volume 32, pages 14014–14024. Curran Associates, Inc.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on*

Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

T. Verelst and T. Tuytelaars. 2020. Dynamic convolutions: Exploiting spatial sparsity for faster inference. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2317–2326.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

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.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. DeeBERT: Dynamic early exiting for accelerating BERT inference. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2246–2251, Online. Association for Computational Linguistics.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. BERT-of-theseus: Compressing BERT by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7859–7869, Online. Association for Computational Linguistics.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*.