AdaPrune: Pruning Transformer with Sparse Regularization

Anonymous ACL submission

Abstract

The key component of transformer architecture is the multi-head self-attention(MHA) and feed forward neural network (FFN). In this paper, we observe that, across many applications, MHA component is *nonsymmetric* and FFN component is *sparse* within transformer architecture. Leveraging this observation, we propose a new method, AdaPrune, to utilize sparse regularization to conduct structure pruning in MHA and FFN modules. This method selects task-specific valuable heads in multi-head attention modules and effective blocks in feed forward layers during the fine-tuning stage, while maintaining the original performance of full transformer model. Extensive experiments show that AdaPrune can achieve competitive performance on these tasks while significantly reduce the computation cost.

1 Introductions

Transformer models (Vaswani et al., 2017) have proven its overwhelming performance among various natural language processing (NLP) problems such as neural machine translation (Liu et al., 2020; Ott et al., 2018), natural language understanding (Devlin et al., 2019; Liu et al., 2019) and automatic speech recognition (Wang et al., 2020a). Despite becoming the fundamental architecture for many applications, transformer model is hard to deploy in practice due to the high computation cost. This is attributed to the multi-head self-attention module (MHA) and feed-forward neural network (FFN) inside the standard transformer architecture (Child et al., 2019; Wu et al., 2020). On the one hand, MHA has quadratic complexity in compute and memory with respect to sequence length of inputs (Vaswani et al., 2017). On the other hand, FFN contributes most of computation FLOPS as it normally consists of a large hidden dimensions (e.g. FFN contains more than 50% of parameters for GPT-3 model families (Brown et al., 2020)).

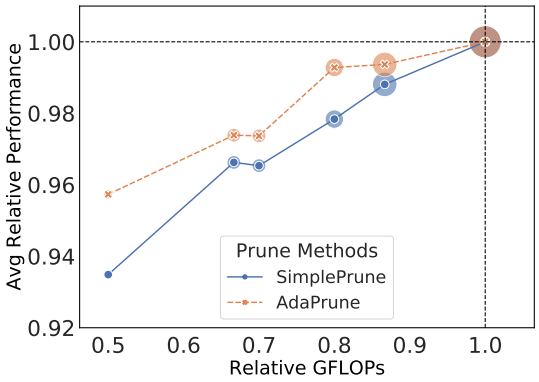

Figure 1: Trade-off illustration between performance (Y-axis) and inference cost (X-axis) of Standard Prune and AdaPrune for RoBERTa base model.

There are several lines of research work focusing on improving the efficiency of transformer architecture: (i) optimizing self-attention mechanism (Tay et al., 2020) such as introducing sparsity (Child et al., 2019) and low-rank projection (Wang et al., 2020b; Choromanski et al., 2020) into attention computation; (ii) knowledge distillation from large pre-trained transformer models to smaller counterpart such as DistilBERT (Sanh et al., 2020); (iii) mixed precision such as training and deploying quantized model (Ott et al., 2019); (iv) model pruning such as removing some heavy component of the transformer model to speed up the inference (Frankle and Carbin, 2019; Chen et al., 2020).

In this paper, we propose a new angle for improving transformer efficiency. We first provide an analysis across a range of NLP classification and translation tasks, to show that, in the standard transformer architecture, (i) if we define the self-attention mechanism as a functional mapping, it is *nonsymmetrical* across each layer and each heads. (ii) feed forward neural network is *extremely sparse*. In details, we observe that more than 97% tensors from FFN is equal to or proximate to zero after the activation.

Leveraging this observation, we proposed a new

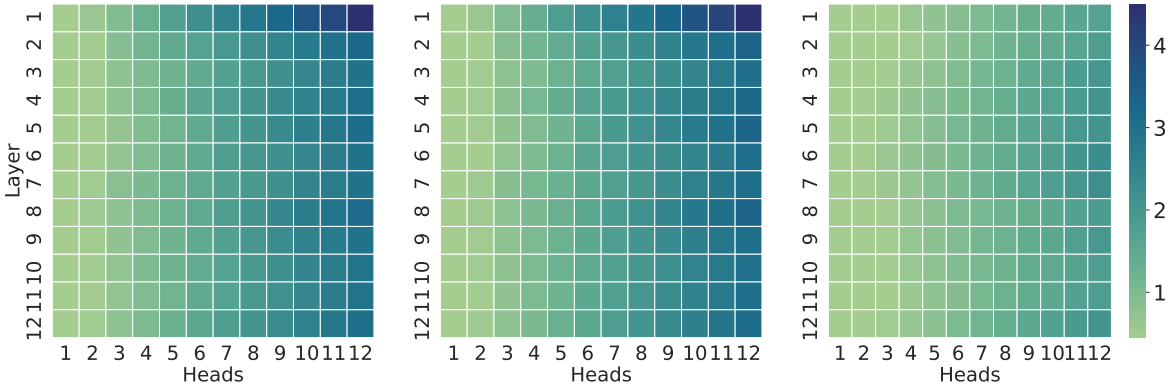

Figure 2: From left to right, we plot the heatmap of spectral norm of $\mathbf{W_i^q}$, $\mathbf{W_i^k}$ and $\mathbf{W_i^v}$ among 144 heads across 12 layers. The results are based on fine-tuned RoBERTa-base model in SST-2 benchmark.

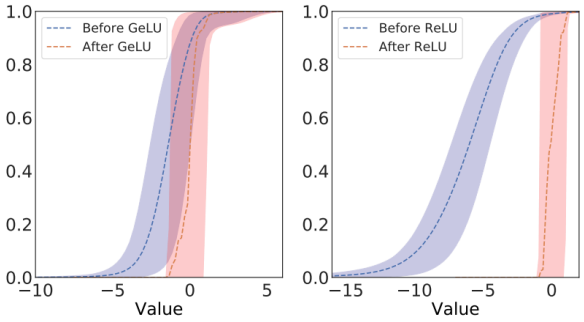

Figure 3: Tensor distribution shift after GeLU/ReLU activation in FFN. The left is based on fine-tuned RoBERTa model on IMBD dataset and the right is based on transformer model on WMT16 en-de translation task.

method to improve transformer efficiency, named as AdaPrune, which utilizes sparse regularization to conduct structured pruning. Our method consists of three steps as described in Section 3: fine-tuning with sparse regularization on in-domain task, prune model (not fine-tuned) based on component rank in MHA and FFN obtained from first step, and fine-tune pruned model on the same target task. We apply our methodology on pre-trained language model such as RoBERTa (Liu et al., 2019) and investigate the performance-efficiency trade off on GLUE benchmark. As shown in Fig. 1, AdaPrune provides average 2% improvement in GLUE benchmark compared to the simple structured prune, which remove each heads and FFN blocks randomly. We believe this method is complementary and can be combined with the existing various transformer efficiency techniques.

2 Motivation

In this section, we will introduce the standard transformer architecture and two important observations inside the multi-head attention and feed forward neural network inside transformer.

2.1 Transformer Architecture

Each transformer layer (Vaswani et al., 2017) includes two important module: MHA and FFN. The MHA can be described as

$$\mathrm{MHA}_i(\mathbf{X}_i) = \mathrm{Concat}[\mathrm{head}_i(\mathbf{X}_i)]_{i=1}^{H}\mathbf{W_{out}},$$

$$\mathrm{head}_i = \mathrm{softmax}\left(\frac{\mathbf{W}_i^q\mathbf{X}_i(\mathbf{W}_i^k\mathbf{X}_i)^T}{d_k}\right)\mathbf{W}_i^v\mathbf{X}_i,$$

where $\mathbf{X}_i$ is the hidden representations in ith layer, H is number of heads in each layer. The query, key and value are produced via linear mapping $\mathbf{W}_i^q$, $\mathbf{W}_i^k$ and $\mathbf{W}_i^v$. The FFN is defined as

$$\mathbf{Y}_i = \mathrm{ReLU}(\mathbf{W}_{\mathrm{fc1},i}\mathrm{MHA}_i(\mathbf{X}_i) + \mathrm{b}_{\mathrm{fc1},i}),$$

$$\mathbf{C}_i = \mathbf{W}_{\mathrm{fc2},i}\mathbf{Y}_i + \mathrm{b}_{\mathrm{fc2},i},$$

where $\mathbf{W}_{\mathrm{fc1},i}, \mathbf{W}_{\mathrm{fc2},i}{}' \in \mathbb{R}^{d\times d_{\mathrm{hidden}}}$. In the sequel, we will omit the subscript i.

2.2 Sparsity and Non-symmetry

Existing work (Tsai et al., 2020) shows that MHA and FFN are the most time consuming modules in transformer architecture, which contributes more than 95% FLOPS. Therefore, we investigate the characteristics of MHA and FFN in each transformer layer to see if we can prune the architecture into a compact size. In MHA, we define the model parameters $\mathbf{W_i^q}$, $\mathbf{W_i^k}$ and $\mathbf{W_i^v}$ as functional mappings and investigate the distribution of spectral norm across different heads and layers. For example, we fine-tune RoBERTa model (Liu et al., 2019) on the Stanford Sentiment Treebank

Binary task (Socher et al., 2013). Then we extract weights from the MHA module as shown in Fig. 2. The heatmap reveals that, the spectral norm of $\mathbf{W_i^q}$, $\mathbf{W_i^k}$ and $\mathbf{W_i^v}$ varies across different heads and different layers. We hypothesis that this phenomenon is due to different token distribution of input instances across different tasks. It provides the possibility to remove the task-specific heads with smaller spectral norm.

Furthermore, we investigate the sparsity of the output from FFN across all layers. In short, there are two linear layers in each transformer layer $\mathbf{W}_{\text{fc1}}$ and $\mathbf{W}_{\text{fc2}}$. Usually, activation (e.g. GeLU/ReLU) is applied to the output from $\mathbf{Y}$. We explore the distribution shift in both classification task (GeLU is applied) and translation task (ReLU is applied). Fig. 3 illustrates the distribution change of the output from $\mathbf{W}_{\text{fc1}}$ before and after activation. For example, in the translation task, we observe more than 97% values after ReLU activation are zero. This implies corresponding linear projections from $\mathbf{W}_{\text{fc1}}$ for these negative outcome values are redundant and can be further removed.

3 Methodologies

In this section, we present the main algorithmic component of AdaPrune, a sparse regularization-based structure pruning strategy. It compresses the transformer architecture through attention heads pruning and block pruning in feed forward networks, and takes three steps to compress the transformer architecture. The main procedure is listed in Algorithm 1.

Fine-tuning with Sparse Regularization: We first add sparse regularization to task loss (e.g. cross-entropy loss in clasification task) during the fine-tuning stage. Suppose we have L layers, the loss is defined as

$$\text{Loss} = L_{\text{task}} + \lambda \left(\sum_{l=1}^{L} \|\text{MHA}_l\| + \|\text{FFN}_l\| \right)$$

$$\|\text{MHA}_l\| = \sum_{i=1}^{H} \|\mathbf{W}_i^q\| + \|\mathbf{W}_i^k\| + \|\mathbf{W}_i^v\|$$

In the above, sparse regularization is added across each layer and each head with a single scaling factor λ. Note that we partition the matrix $\mathbf{W}_{\text{fc1}}$ and $\mathbf{W}_{\text{fc2}}$ of FFN into $M \times N$ blocks, and calculate the norm of each block separately. Then the norm of FFN in transformer layer l is defined as

$$\|\text{FFN}_l\| = \sum_{i=1}^{M} \sum_{j=1}^{N} \|\mathbf{W}_{\text{fc1},i,j}\| + \|\mathbf{W}_{\text{fc2},i,j}\|$$

Algorithm 1 AdaPrune

procedure ADAPRUNE(D_{train}, M)
  M_T = **AdaTrain**(D_{train}, **M**, λ)
  $\mathbf{M}^{pruned}$ = **Prune**(M_T, **M**, ϵ_m, ϵ_f)
  for *epoch* in *epochs* **do**
    for X_{train_i}, Y_{train_i} in D_{train} **do**
      pred = $\mathbf{M}^{pruned}(X_{train_i})$
      loss = loss(pred, Y_{train_i})
      Backpropagation(loss)
    end for
  end for
  return $\mathbf{M}_T^{pruned}$
end procedure

Norm-based Pruning: After fine-tuning with sparse regularization, we use the norm $\|\mathbf{W}_i^q\| + \|\mathbf{W}_i^k\| + \|\mathbf{W}_i^v\|$ of each head as the proxy for head importance, and remove the unimportant heads based on the pre-defined norm threshold ϵ_m. Similarly, importance of blocks in feed forward networks are calculated based on the norm of corresponding blocks, and unimportant blocks are removed based on threshold ϵ_f. As a result, we could control the strength of pruning by simply tuning the thresholds ϵ_m and ϵ_f. We will store the pruning information in set M_T.

Fine-tune Pruned Model: Based on head and block index stored in M_T, we will prune pre-trained language model $\mathbf{M}$ to obtain a smaller pre-trained language model $\mathbf{M}^{pruned}$. Then we further fine-tune the pruned model $\mathbf{M}^{pruned}$ on the same target tasks.

4 Experiments

In our experiments, we use the RoBERTa-base model (Liu et al., 2019) and evaluate proposed method on various tasks with three pruning settings: MHA pruning, FFN pruning and combination of the MHA & FFN pruning. For each setting, we have explored two pruning percentages (33% or 50%), which means that we remove 33% or 50% parameters in the corresponding module. We evaluate the performance of AdaPrune in the GLUE benchmark (Wang et al., 2019). We also compare our proposed method with two baseline methods:

Method	**IMDB** (Acc)	**SST-2** (Acc)	**CoLA** (Acc)	**MNLI** (Acc)	**QNLI** (Acc)	**QQP** (Acc)	**STS** (Pear./Sp.)	**AVG**
Standard Prune								
H=8, D=3072	92.7	93.5	83.6	86.2	91.7	91.5	89.4/89.1	89.7
H=6, D=3072	91.9	93.2	82.2	85.7	90.9	91.4	88.8/88.5	89.1
H=12, D=2048	91.8	93.1	78.1	86.4	91.8	91.3	89.5/89.1	88.9
H=12, D=1536	91.2	92.1	73.6	85.0	90.1	91.1	89.6/89.3	87.8
H=8, D=2048	90.8	92.3	77.5	84.7	90.0	91.1	88.1/87.8	87.8
H=6, D=1536	89.3	90.6	71.5	83.0	87.4	90.6	83.7/83.8	85.0
AdaPrune								
H=8, D=3072	93.0	95.1	83.9	86.8	92.1	91.6	89.8/89.5	90.2
H=6, D=3072	92.0	94.5	82.9	86.0	91.3	91.5	89.2/88.9	89.5
H=12, D=2048	92.9	94.5	83.2	87.2	92.6	91.8	89.6/89.4	90.1
H=12, D=1536	91.2	93.5	77.9	85.8	90.9	91.3	88.7/88.4	88.5
H=8, D=2048	91.3	93.8	80.7	86.0	91.6	91.6	86.4/86.2	88.5
H=6, D=1536	88.9	93.0	77.1	84.8	90.0	91.4	85.4/85.2	87.0
DistilBERT (Sanh et al., 2020)	92.8	91.3	51.3	82.2	89.2	88.5	84.5/85.0	83.1
RoBERTa (Liu et al., 2019)	93.7	95.1	83.1	87.5	93	91.9	91.3/90.9	90.8

Table 1: Results of the RoBERTa-base model with different pruning methods. The pruning configuration includes the number of heads in MHA of each layer and the dimension of $\mathbf{W}_{\text{fc1}}$ in the FFN. For example, H=12 heads means there are 12 heads in each transformer layer and D=3072 indicate the two feed forward layers are of size [768, 3072] and [3072, 768].

(i) **Standard Prune**: randomly remove the heads and FFN blocks; (ii) **Distill BERT** (Sanh et al., 2020): distill the pre-trained language model into a 6-layer transformer model. As shown in the table 1, our proposed method, AdaPrune performs consistently better than Standard Prunes DistillBERT across all tasks with different pruning setup. For example, If we remove 33% FFN blocks, standard pruning method shows 2.16% relative drop while our proposed method only shows 0.73% relative drop on average. If we remove 33% heads in MHA, we observe 1.27% relative drop with Standard Prune while our proposed method have only 0.75% relative drop.

Setup	GFLOPs	Time (ms)	Savings
H=12, D=3072	96.6	809	/
H=8, D=3072	83.6	685	15.3%
H=6, D=3072	77.3	613	24.2%
H=12, D=2048	77.2	653	19.3%
H=12, D=1536	67.6	570	29.5%
H=8, D=2048	64.4	553	31.6%
H=6, D=1536	48.3	420	51.9%

Table 2: Summary of efficiency improvement with different pruning configurations in MHA and FFN. The first line is the baseline coming from the RoBERTa base model without pruning. Since GFLOPs depend on the size of the input feature, GFLOPs in the table are calculated based on the input tensor with batch size as 1, sequence length as 512 and embedding size as 768.

Furthermore, we benchmark the efficiency improvement for the prune model. We harvest the inference time based on Intel(R) Xeon(R) Platinum 8259CL CPU@2.50GHz for all kinds of parameter settings used in experiments. For each pruning setup, we pass the same test example with sequence length 512 as input for models and take the average inference time on 50 runs. The Table 2 shows the speed up varies from 15.36% to 48.09% with different prune setup.

5 Conclusion

In this work, we study the AdaPrune strategy with the goal of achieving both efficiency and effectiveness. We start with analysis on MHA and FFN modules to show our observation of weights distribution on these modules. Based on observation, we find that heads in MHA have different weights scale and we take that as indicator for importance. Additionally, we find output value from FC1 are cropped sharply at zero due to GeLU activation, which implies the unnecessary calculation exists in feed forward layers. Through experiments of applying AdaPrune to RoBERTa model, we prove the decent performance of AdaPrune against Standard Prune strategy.

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.

Xiaohan Chen, Yu Cheng, Shuohang Wang, Zhe Gan, Zhangyang Wang, and Jingjing Liu. 2020. Earlybert: Efficient bert training via early-bird lottery tickets. *arXiv preprint arXiv:2101.00063*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks.

Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. 2020. Very deep transformers for neural machine translation.

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.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *WMT*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*.

Henry Tsai, Jayden Ooi, Chun-Sung Ferng, Hyung Won Chung, and Jason Riesa. 2020. Finding fast transformers: One-shot neural architecture search by component composition.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, page 5998–6008.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Changhan Wang, Yun Tang, Xutai Ma, Anne Wu, Dmytro Okhonko, and Juan Pino. 2020a. fairseq s2t: Fast speech-to-text modeling with fairseq.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020b. Linformer: Self-attention with linear complexity.

Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. 2020. Lite transformer with long-short range attention.