DAP-BERT: Differentiable Architecture Pruning of BERT

Chung-Yiu Yau(✉), Haoli Bai, Irwin King, and Michael R. Lyu

The Chinese University of Hong Kong, Shatin, Hong Kong, China
1155109029@link.cuhk.edu.hk, {hlbai,king,lyu}@cse.cuhk.edu.hk

Abstract. The recent development of pre-trained language models (PLMs) like BERT suffers from increasing computational and memory overhead. In this paper, we focus on automatic pruning for efficient BERT architectures on natural language understanding tasks. Specifically, we propose differentiable architecture pruning (DAP) to prune redundant attention heads and hidden dimensions in BERT, which benefits both from network pruning and neural architecture search. Meanwhile, DAP can adjust itself to deploy the pruned BERT on various edge devices with different resource constraints. Empirical results show that the $\text{BERT}_{\text{BASE}}$ architecture pruned by DAP achieves 5× speed-up with only a minor performance drop. The code is available at https://github.com/OscarYau525/DAP-BERT.

Keywords: Natural language processing · Neural architecture search · Pruning · BERT

1 Introduction

In the study of natural language processing (NLP), pre-trained language models (PLMs) have shown strong generalization power on NLP tasks [7,15]. However, the high computational overhead and memory consumption of these PLMs prohibit the deployment of these models on resource-limited devices, and thus motivates various efforts towards network compression on PLMs. This area has been investigated by numerous studies, such as pruning [6,16,17], knowledge distillation [11,12], quantization [1,3,23,33].

Among these methods, network pruning starts from a pre-defined architecture and simplifies it by removing unimportant parameters in the network. However, most existing pruning methods rely on hand-crafted criteria to decide the sub-network structure, such as the magnitude of parameters [6] or its gradients [20]. On the other hand, neural architecture search (NAS) aims to automatically search for optimal network architectures and avoids human intervention at the stage of architecture design. Despite the success of NAS popularized in convolution neural networks and recurrent neural networks [8,14], little work has been put on applying NAS to attention-based Transformer networks such as BERT. This is due to the expensive pre-training of language models, which makes the searching process quite time-consuming. While there are some works

© Springer Nature Switzerland AG 2021
T. Mantoro et al. (Eds.): ICONIP 2021, LNCS 13108, pp. 367–378, 2021.
https://doi.org/10.1007/978-3-030-92185-9_30

that search architecture during the fine-tuning stage [5,16], they either suffer from computationally expensive CNN-based cells [5] or inaccurate control of model size through sparse regularization [16].

In this paper, we propose differentiable architecture pruning (DAP) for BERT, a novel approach that benefits from both BERT pruning and neural architecture search. Specifically, our proposed DAP can automatically discover the optimal head number for self-attention and the dimensionality for the feed-forward networks given the resource constraints from various edge devices. Inspired by [14], we assign each architecture choice with learnable parameters, which can be updated by end-to-end training. Meanwhile, to stabilize the searching algorithm, we further introduce rectified gradient update for architecture parameters, as well as progressive architecture constraint, such that the searching process can proceed smoothly. Finally, to find a sub-architecture that performs comparable to the original network, we apply knowledge distillation as a clue to architecture searching and model re-training so that the sub-network mimics the behaviours of its original network.

We conduct extensive experiments and discussions on the GLUE benchmark to verify the proposed approach. The empirical results show that the inference time of our pruned $\text{BERT}_{\text{BASE}}$ can be accelerated up to 5× with only a minor performance drop. Moreover, to the limit of compression, our pruned model can reach up to 27× inference speedup compared with the original $\text{BERT}_{\text{BASE}}$ model, while maintaining 95% of its average performance over GLUE tasks.

2 Related Work

Neural network pruning and neural architecture search are both rapidly growing fields with a large amount of literature. We summarize these two strands of research that are closely related to our proposed solution in the following sections.

2.1 Network Pruning for BERT

Network pruning aims to remove the unnecessary connections in the neural network [2,9,28,30]. Gordon et al. [10] investigate the effect of weight magnitude pruning during the pre-training stage on the transferability to downstream tasks. Prasanna et al. [20] apply gradient-informed structured pruning and unstructured weight magnitude pruning on BERT to verify the lottery ticket hypothesis [9], which indirectly points out the ineffectiveness of sensitivity-based structured pruning. McCarley et al. [16] incorporate distillation and structured pruning by L_0 regularization. Our proposed structured pruning method differs from the usual sensitivity-based approaches [16,20] or weight magnitude approaches [6,10] since we avoid hand-crafted criteria on pruning. Instead, we adopt a loss objective that accurately reflects the model FLOPs constraint and the prediction behaviour of the original model.

2.2 Neural Architecture Search

Neural architecture search (NAS) is to automatize the design of neural network architecture [14,27,35]. Expensive approaches such as reinforcement learning [35] and evolutionary algorithm [21] spend thousands of GPU days to obtain the optimal architecture for computer vision tasks. Recent efforts such as DARTS [14] follow differentiable architecture search, which adopt continuous relaxations over all possible operations in the search space for gradient-based optimization. Such NAS techniques can also be applied for network compression such as pruning [8,31] and quantization [13,29,31]. Especially, TAS [8] search for the optimal width and depth of CNN-based networks in a similar differentiable fashion and achieve effective network pruning. For BERT architecture, AdaBERT [5] performs differentiable cell-based NAS and searches for a novel small architecture by task-oriented knowledge distillation, and obtains efficient CNN-based models. Recently, NAS-BERT [32] performs block-wise NAS with knowledge distillation during the pre-training stage. However, these approaches search the architectures in the search space from scratch, which can be slow in practice. In this paper, instead of searching for a novel architecture from scratch, we start from existing BERT architectures. We follow the differentiable architecture search in a super-graph defined by the original BERT model to prune potentially ineffective connections.

3 Methodology

In this section, we present differentiable architecture pruning (DAP) for BERT, an automatic pruning solution that can be tailored for various edge devices with different resource constraints. The proposed algorithm is initialized with a trained PLM for the downstream task, which avoids the time-consuming pre-training of PLMs. An overview of our searching approach is illustrated in Fig. 1.

3.1 Definition of Search Space

We aim to find the best layer-wise configuration of heads in multi-head attention (MHA) and intermediate dimensionality in feed-forward network (FFN) of the transformer. The search space of MHA and FFN is designed as follows:

Multi-head Attention. Recall that with input X to the MHA, the i-th head H_i can be computed as

$$Q_i = XW_i^Q,\ K_i = XW_i^K,\ V_i = XW_i^V,\ H_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d}}\right) V_i, \quad (1)$$

where W_i^Q, W_i^K, W_i^V are projection matrices of query, key and value for the i-th head respectively, and d is the head size. To prune away unnecessary heads, we assign $\alpha^m \in \mathbb{R}^N$ as the architecture parameter for each head, where N is the

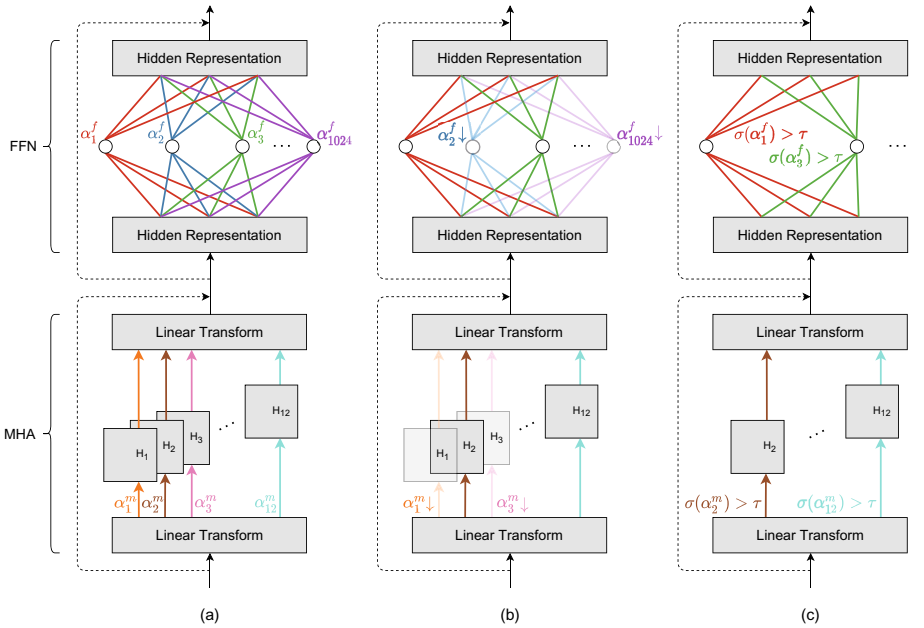

Fig. 1. An overview of the search method and search space of a BERT hidden layer, with multi-head attention block in the bottom and feed-forward block on the top. Dashed arrows represent skip connections. (a) The original network is initialized as the super-graph. An architecture parameter α_i is assigned to each group of weights of the same color in the diagram. (b) Learning the α_i w.r.t. the objective function. (c) Output the optimal sub-network by selecting α that exceeds the threshold τ.

total head numbers before pruning. We further assign a sigmoid function that ensures $\sigma(\alpha^m) \in [0,1]^N$ for head selection. Thus, the weighted MHA output can be written as

$$\text{MultiHead}(Q,K,V) = \text{Concat}(H_1 \odot \sigma(\alpha_1^m),\ ...,\ H_N \odot \sigma(\alpha_N^m))\ W^O. \tag{2}$$

Whenever $\sigma(\alpha_i^m) \to 0$, the i-th head can be safely pruned without affecting the output. Note that for each pruned head, the dimensionality of the projection matrices W^Q, W^K, W^V and W^O will be adjusted accordingly.

Feed-Forward Network. The feed-forward network is composed of an intermediate layer followed by an output layer. We aim at reducing the dimensionality of the intermediate representation, by introducing architecture parameters $\alpha^f \in \mathbb{R}^D$ for the D intermediate dimensions. The FFN output can thus be written as

$$\text{FFN}(X) = \max(0, XW_1 + b_1)\ \text{Diag}\big(\sigma(\alpha^f)\big)\ W_2 + b_2, \tag{3}$$

where X is the input to FFN, $Diag(\cdot)$ represents the diagonal matrix, and W_1, W_2, b_1, b_2 are the parameters of the two linear layers.

3.2 Differentiable Architecture Pruning

Given the architecture parameters associated with the search space, we follow differentiable architecture search [14] to update α via end-to-end training. In order to find the best architecture on edge devices with different computational capacities, we also include the network FLOPs as part of the search objective. The overall searching problem can be formulated as a bi-level optimization problem as follows:

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}(w^*(\alpha), \alpha) \\ \text{subject to} \quad & w^*(\alpha) = \text{argmin}_w \ \mathcal{L}(w, \alpha), \\ & \mathcal{F}(\alpha) \leq \mathcal{F}_{\text{target}}, \end{aligned} \tag{4}$$

where $\mathcal{F}(\alpha)$ denotes the number of FLOPs based on the architecture α, and $\mathcal{F}_{\text{target}}$ is the searching target FLOPs. To satisfy the FLOPs constraint in Eq. (4), we follow [8] to apply FLOPs penalty as a differentiable loss objective $\mathcal{L}_{\text{cost}}$ w.r.t. α as follows:

$$\mathcal{L}_{\text{cost}} = \begin{cases} \log(\mathbb{E}[\mathcal{F}(\alpha)]) & \text{if } \mathbb{E}[\mathcal{F}(\alpha)] > (1+\delta) \times \mathcal{F}_{\text{curr}}(t), \\ 0 & \text{if } (1-\delta) \times \mathcal{F}_{\text{curr}}(t) \leq \mathbb{E}[\mathcal{F}(\alpha)] \leq (1+\delta) \times \mathcal{F}_{\text{curr}}(t), \\ -\log(\mathbb{E}[\mathcal{F}(\alpha)]) & \text{if } \mathbb{E}[\mathcal{F}(\alpha)] < (1-\delta) \times \mathcal{F}_{\text{curr}}(t), \end{cases} \tag{5}$$

$$\mathbb{E}[\mathcal{F}(\alpha)] = \sum_{l=1}^{L} \left(\sum_{j=1}^{H} \sigma(\alpha_{lj}^m) \mathcal{F}_{\text{MHA}} + \sum_{j=1}^{D} \sigma(\alpha_{lj}^f) \mathcal{F}_{\text{FFN}} \right), \tag{6}$$

where $\mathbb{E}[\mathcal{F}(\alpha)]$ is the expected FLOPs of the current architecture summed over L hidden layers, $\mathcal{F}_{\text{MHA}}$ and $\mathcal{F}_{\text{FFN}}$ are the FLOPs of a single head and one intermediate dimension in FFN, and δ is a tolerance parameter. $\mathcal{F}_{\text{curr}}(t)$ denotes the target FLOPs, which is time-dependent as will be introduced in Eq. (8).

Rectified Update of $\boldsymbol{\alpha}$. During architecture search, we optimize both w and α by stochastic gradient descent. However, the update of α suffers from vanishing gradient as a result of the sigmoid activation $\sigma(\cdot)$. To solve this challenge, we introduce rectified update for α as follows. We adopt sign stochastic gradient descent (signSGD) [4] to enlarge the magnitude of gradients on α. It is known that signSGD avoids the problem of vanishing gradient since the magnitude of gradient is controlled [18]. Nevertheless, signSGD may bring oscillations that make the optimization unstable. These rapid oscillations to the architecture parameter α may lead to an immature solution. To smoothly stabilize the optimization process, we only backpropagate the top-10% gradient of α according to their magnitude within the search region, while the rest are masked as follows:

$$\hat{g}_i^{(t)} = \begin{cases} \text{sign}(g_i^{(t)}) & \text{if } g_i^{(t)} \text{ is top-10\% in magnitude,} \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Progressive Architecture Constraint. Directly applying a FLOPs penalty with a fixed FLOPs target leads α to arrive at the target size early during

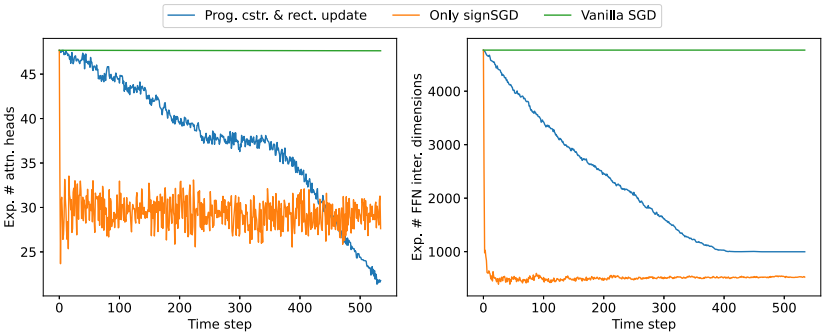

Fig. 2. The searching dynamics of different approaches. (**Left**) shows the expected number of attention heads. (**Right**) shows the FFN intermediate dimensions.

searching, while progressive pruning usually better identifies the less influential parameters [6]. To perform architecture search progressively, we adjust the FLOPs target $\mathcal{F}_{\text{curr}}(t)$ at time step t as

$$\mathcal{F}_{\text{curr}}(t) = \mathcal{F}_{\text{original}} \exp\left(\frac{1}{T} \ln \frac{\mathcal{F}_{\text{target}}}{\mathcal{F}_{\text{original}}}\right)^t, \tag{8}$$

where T is the scheduled number of training steps, $\mathcal{F}_{\text{original}}$ is the original architecture FLOPs, and $\mathcal{F}_{\text{target}}$ is the desired FLOPs.

We visualize the effect of rectified update and progressive architecture constraint in Fig. 2. It is evident that vanilla SGD suffers from vanishing gradient, thus cannot achieve pruning. SignSGD directly arrives at the FLOPs target in the beginning, which may result in sub-optimal architectures. On the other hand, our progressive FLOPs constraint and rectified update enable smooth searching.

Loss Objective for Searching. Due to the intractability of the bi-level problem in Eq. (4), we simultaneously update w and α w.r.t. the objective function on the training set. The objective function involves two terms: the cross-entropy between the full-size model logits z^t and the searching model logits z^s (i.e., knowledge distillation); and the FLOPs penalty $\mathcal{L}_{\text{cost}}$ as defined in Eq. (5). The searching objective is thus

$$\mathcal{L}_{\text{search}} = \mathcal{L}_{\text{ce}} + \lambda \mathcal{L}_{\text{cost}}, \quad \text{where } \mathcal{L}_{\text{ce}} = -\operatorname{softmax}(z^t) \cdot \operatorname{log\_softmax}(z^s). \tag{9}$$

After the training of α, pruning is achieved with a pre-set threshold τ, i.e., only keeping the connections that satisfy $\sigma(\alpha_i) > \tau$.

3.3 Fine-Tuning with Two-Stage Knowledge Distillation

After obtaining the slimmed BERT structure, we further fine-tune the network to recover from performance degradation due to pruning. The fine-tuning is based

Algorithm 1: Learning optimal sub-network of BERT

Initialize network weights w from a well-trained model;
Initialize architecture parameters α that satisfy $\sigma(\alpha_i) > \tau$;
▷ *Searching*;
for T *iterations* **do**
  Forward pass to compute MHA and FFN by Eq. (2), (3);
  Update $\mathcal{F}_{\text{curr}}(t)$ according to Eq. (8);
  Calculate the loss objective in Eq.(9);
  Backpropagate to update w by Adam optimizer;
  Backpropagate to update α with rectified gradient in Eq. (7);
end
▷ *Pruning*;
Prune by selecting $\sigma(\alpha_i) > \tau$;
▷ *Fine-tuning*;
Restore network parameter value from the original network;
Fine-tune the pruned network with two-stage knowledge distillation;

on two-stage knowledge distillation [12] given its previous success in model compression. The first stage aims at intermediate layer distillation, which minimizes the mean squared error (MSE) between the sub-network (student) and the original network (teacher) as follows:

$$\mathcal{L}_{\text{int}} = \sum_{l=1}^{L} \|A_l^s - A_l^t\|_F^2 + \|F_l^s - F_l^t\|_F^2, \tag{10}$$

where $\|\cdot\|_F$ is the Frobenius norm, A_l^s, A_l^t and F_l^s, F_l^t denote the MHA attention maps and FFN output of l-th transformer layer from the student and teacher model, respectively. Note that as we do not modify the output shape of the student model, the MSE loss can be directly calculated without linear mapping to align the dimension as done in [12]. The second stage is prediction layer distillation, which similarly adopts the cross-entropy loss $\mathcal{L}_{ce}$ defined in Eq. (9).

In practice, we find that fine-tuning the network parameters immediately after searching usually lead to sub-optimal performance. Similar observations are also found in existing NAS literature [14,19,27], where instead they initialize the network parameters and train the architecture from scratch. Similarly, we only inherit the network structure (i.e., the configuration of heads and dimensions in MHA and FFN), and restore the parameters from the original model.

3.4 Summary of the Method

Algorithm 1 summarizes the overall workflow of our proposed method, which can be generally divided into three steps: searching, pruning and fine-tuning. Firstly, we initialize all architecture parameters by $\sigma(\alpha_i) > \tau$ such that all the prunable MHA heads and FFN dimensions are kept initially. We conduct searching by simultaneously minimizing the objective function in Eq. (9) w.r.t. architecture

parameter α and network parameters w. To facilitate a smooth searching process, we rectify the update of α according to Eq. (7), and incorporate progressive architecture constraint in Eq. (8). After searching, we apply pruning based on α and restore the network parameters to their original states. Finally, we conduct two-stage knowledge distillation for fine-tuning.

4 Experiments

In this section, we empirically verify the proposed method on the GLUE benchmark [26]. We first introduce the experiment setup in Sect. 4.1. The main results are presented in Sect. 4.2, followed by comparisons with other state-of-the-art approaches in Sect. 4.3. Finally, we provide further discussions to better understand the proposed approach in Sect. 4.4.

4.1 Experiment Setup

Dataset and Metrics. The GLUE benchmark provides a variety of natural language understanding tasks. Unless specified, we report the metrics of each task as follows: Matthew's Correlation for CoLA, F1 score for MRPC and QQP, Spearman Correlation for STS-B, and accuracy for the remaining tasks. Following [12], We apply data augmentation to small datasets (RTE, MRPC, STS-B and CoLA) to improve fine-tuning of the pruned networks.

Implementation. The proposed method applies to any well-trained BERT models on downstream tasks. We take the $\text{BERT}_{\text{BASE}}$ [7][1] and TinyBERT [12][2] as the super-graph for searching. For each of the super-graphs, we experiment with different FLOPs constraint $\mathcal{F}_{\text{target}}$ and compare the performance drop across different models. For all the experiments, we initialize $\alpha_i = 5$ and use $\tau = 0.99$ as the pruning threshold. For small datasets, we search for 10 epochs and fine-tune for 10 epochs. For large datasets, we search for one or fewer epochs (i.e., using part of the training set) and fine-tune for 3 epochs. Then we fine-tune the sub-network using Adam optimizer with 5×10^{-5} learning rate.

4.2 Experiment Results

We evaluate DAP on $\text{BERT}_{\text{BASE}}$, TinyBERT_4, and TinyBERT_6, and results are shown in Table 1. We denote our results as +DAP-p%, where p% denotes the pruning rate. It can be found that the accuracy drop depends on the original network size, where on the same scale of FLOPs reduction, small networks bear

[1] To obtain the task-specific parameters, we follow the standard fine-tuning pipeline in https://huggingface.co/bert-base-uncased.

[2] Task specific model parameters available at https://github.com/huawei-noah/Pretrained-Language-Model/tree/master/TinyBERT.

a larger percentage of accuracy drop than large networks. Notably, $\text{BERT}_{\text{BASE}}$ can be pruned to half without accuracy degradation.

Additionally, we also measure the practical inference speedup of the pruned networks in Table 2. It is shown that FLOPs reduction on the network architecture can bring up to 5.4× practical speed-up for $\text{BERT}_{\text{BASE}}$, and can even be 27.6× faster for TinyBERT_4 with DAP-30%.

Table 1. Experimental results of the proposed architecture searching algorithm, evaluated on the GLUE test set.

#	Models	FLOPs (B)	Param. (M)	MNLI-m 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Avg. (%↓)
1	$\text{BERT}_{\text{BASE}}$	22.3	109.5	84.0	70.7	91.1	92.7	55.3	82.5	86.6	65.5	78.6 (0.0)
2	+ DAP-50%	11.4	66.6	84.2	72.2	90.4	93.2	53.0	83.0	86.6	66.0	78.6 (0.0)
3	+ DAP-30%	7.1	51.6	83.6	71.6	89.9	91.9	49.9	82.6	86.5	65.2	77.7 (1.1)
4	+ DAP-10%	3.0	33.6	83.0	71.4	88.3	91.8	45.7	81.7	85.8	63.6	76.4 (2.7)
5	TinyBERT_6	11.1	67.0	84.6	71.6	90.4	93.1	51.1	83.7	87.3	70.0	79.0 (0.0)
6	+ DAP-30%	3.6	37.1	83.7	71.8	89.5	93.0	46.1	83.3	86.9	63.6	77.2 (2.2)
7	TinyBERT_4	1.2	14.5	82.5	71.3	87.7	92.6	44.1	80.4	86.4	66.6	76.5 (0.0)
8	+ DAP-30%	0.4	11.1	80.8	70.9	84.4	91.8	40.7	78.6	85.4	60.8	74.0 (3.2)

Table 2. Practical speedup of the sub-networks, presenting the inference time for a batch of 32 examples with 128 maximum sequence length on Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00 GHz with 4 cores.

#	Models	FLOPs (B)	Speedup (×)	Time (s)	Practical Speedup (×)
1	$\text{BERT}_{\text{BASE}}$	22.3	1.0	7.28	1.0
2	+ DAP-50%	11.4	2.0	3.72	2.0
3	+ DAP-30%	7.1	3.1	2.49	2.9
4	+ DAP-10%	3.0	7.3	1.36	5.4
5	TinyBERT_6	11.1	2.0	3.69	2.0
6	+ DAP-30%	3.6	6.2	1.22	6.0
7	TinyBERT_4	1.2	17.9	0.62	11.8
8	+ DAP-30%	0.4	55.4	0.26	27.6

4.3 Comparison with State-of-the-arts

To further validate the proposed approach, we compare with several state-of-the-art compression baselines including vanilla BERT [25], DistilBERT [22], MobileBERT [24], NAS-BERT [32] and Mixed-vocab KD [34]. Evaluations on the GLUE test set and development set are shown in Table 3 and Table 4 respectively. The proposed DAP shows superior performance against the baselines. For instance, our DAP-BERT_{12}-10% achieves the averaged test score of 76.4 with only 3.0 FLOPs (B), which is just 2.2 score lower than the original $\text{BERT}_{\text{BASE}}$ model with more than 7× FLOPs reduction.

4.4 Discussion

Rectified Update and Progressive Architecture Pruning. The left side of Fig. 3 shows the ablation studies for our rectified update and progressive architecture constraint. It can be found that when armed with only progressive constraint, the searching algorithm fails to converge to the desired FLOPs. While pure signSGD can converge to network architectures with desired FLOPs, the performance is usually worse due to oscillating update of α, as previously discussed in Fig. 2. When combined with the rectified update, the performance is consistently improved at different FLOPs targets. Finally, when the rectified update is combined with progressive architecture constraint, the network performance is boosted since the searching dynamics is smooth and stabilized.

Table 3. Comparison with state-of-the-art compression approaches, evaluated on the GLUE test set.

#	Models	FLOPs (B)	Param. (M)	MNLI-m	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
1	$\text{BERT}_{\text{BASE}}$	22.3	109.5	84.0	70.7	91.1	92.7	55.3	82.5	86.6	65.5	78.6
2	$\text{BERT}_{\text{SMALL}}$	3.4	29.2	77.6	68.1	86.4	89.7	27.8	77.0	83.4	61.8	71.5
3	$\text{MobileBERT}_{\text{TINY}}$	3.1	15.1	81.5	68.9	**89.5**	91.7	**46.7**	80.1	**87.9**	**65.1**	**76.4**
4	**DAP-BERT$_{12}$ – 10%**	3.0	33.0	**83.0**	**71.4**	88.3	**91.8**	45.7	**81.7**	85.8	63.6	**76.4**
5	$\text{BERT}_{\text{MINI}}$	0.87	11.1	74.8	66.4	84.1	85.9	0.0	73.3	81.1	57.9	65.4
6	Mixed-vocab KD	-	10.9	80.7	-	-	90.6	-	-	**87.2**	-	-
7	**DAP-BERT$_{4}$ – 30%**	0.40	11.1	**80.8**	**70.9**	**84.4**	**91.8**	**39.6**	**78.6**	85.4	**60.8**	**74.0**

Table 4. Comparison with state-of-the-art compression approaches, evaluated on the GLUE development set.

#	Models	FLOPs (B)	Param. (M)	MNLI-m	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
1	NAS-BERT_{10}	2.3	10.0	76.4	88.5	86.3	88.6	34.0	84.8	79.1	66.6	75.5
2	DistilBERT_{6}	-	66.0	82.2	88.5	**89.2**	91.3	51.3	86.9	**87.5**	59.9	79.6
3	**DAP-BERT$_{12}$ – 10%**	3.0	33.0	**82.8**	**90.6**	88.9	**91.9**	**52.3**	**88.2**	85.3	**67.5**	**80.9**
4	NAS-BERT_{5}	0.86	5.0	74.4	85.8	84.9	87.3	19.8	83.0	79.6	**66.6**	72.7
5	**DAP-BERT$_{4}$ – 30%**	0.40	11.1	**81.2**	**90.6**	**86.2**	**92.2**	**45.8**	**85.8**	**86.0**	63.2	**78.9**

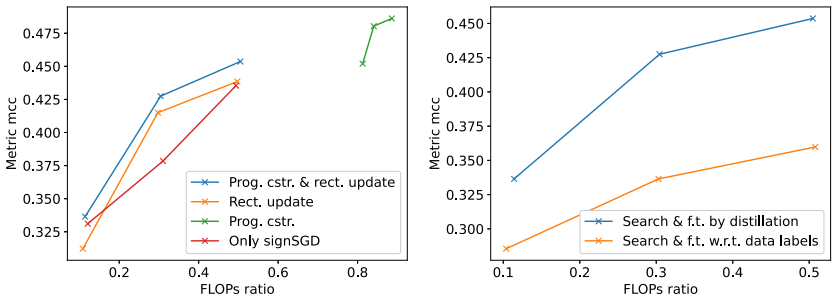

Fig. 3. (**Left**) shows the architecture accuracies under different approaches. (**Right**) shows the effect of knowledge distillation for searching and fine-tuning.

Distillation for Architecture Searching. We verify the advantage of knowledge distillation (soft labels from the original model) over data labels (hard labels from ground truth) by comparing the performance of the architectures found using these search objectives. The empirical result in the right of Fig. 3 shows that knowledge distillation using soft labels can generally find better architectures than using the ground truth data labels.

5 Conclusion

In this paper, we propose differentiable architecture pruning, an automatic neural architecture search for BERT pruning. Given the resource constraints from edge devices, the proposed approach can identify the best model architecture accordingly. Empirical results on the GLUE benchmark show that the pruned BERT model can perform on par with the original network, while enjoying significant inference speedup. Our work opens the door to deploying PLM to resource-limited edge devices and contributes to the various applications of NLP.

Acknowledgement. The work described in this paper was partially supported by the National Key Research and Development Program of China (No. 2018AAA0100204) and the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14210920 of the General Research Fund).

References

1. Bai, H., Hou, L., Shang, L., Jiang, X., King, I., Lyu, M.R.: Towards efficient post-training quantization of pre-trained language models. Preprint arXiv:2109.15082 (2021)
2. Bai, H., Wu, J., King, I., Lyu, M.: Few shot network compression via cross distillation. In: AAAI, vol. 34, pp. 3203–3210 (2020)
3. Bai, H., et al.: BinaryBERT: pushing the limit of BERT quantization. In: ACL (2020)
4. Bernstein, J., Wang, Y.X., Azizzadenesheli, K., Anandkumar, A.: signSGD: Compressed optimisation for non-convex problems. In: ICML (2018)
5. Chen, D., et al.: AdaBERT: task-adaptive BERT compression with differentiable neural architecture search. In: IJCAI (2021)
6. Chen, T., et al.: The lottery ticket hypothesis for pre-trained BERT networks. In: NeurIPS (2020)
7. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT (2019)
8. Dong, X., Yang, Y.: Network pruning via transformable architecture search. In: NeurIPS (2019)
9. Frankle, J., Carbin, M.: The lottery ticket hypothesis: finding sparse, trainable neural networks. In: ICML (2018)
10. Gordon, M.A., Duh, K., Andrews, N.: Compressing BERT: studying the effects of weight pruning on transfer learning. In: ACL (2020)
11. Hou, L., Huang, Z., Shang, L., Jiang, X., Chen, X., Liu, Q.: DynaBERT: dynamic BERT with adaptive width and depth. In: NeurIPS (2020)

12. Jiao, X., et al.: TinyBERT: distilling BERT for natural language understanding. In: EMNLP (2020)
13. Li, Y., Wang, W., Bai, H., Gong, R., Dong, X., Yu, F.: Efficient bitwidth search for practical mixed precision neural network. Preprint arXiv:2003.07577 (2020)
14. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: ICLR (2019)
15. Liu, Y., et al.: RoBERTa: a robustly optimized BERT pretraining approach. Preprint arXiv:1907.11692 (2019)
16. McCarley, J.S., Chakravarti, R., Sil, A.: Structured pruning of a BERT-based question answering model. Preprint arXiv:1910.06360 (2021)
17. Michel, P., Levy, O., Neubig, G.: Are sixteen heads really better than one? In: NeurIPS (2019)
18. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: ICML (2013)
19. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. In: ICML, pp. 4092–4101 (2018)
20. Prasanna, S., Rogers, A., Rumshisky, A.: When BERT plays the lottery, all tickets are winning. In: EMNLP (2020)
21. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: AAAI (2019)
22. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In: NeurIPS (2020)
23. Shen, S., et al.: Q-BERT: hessian based ultra low precision quantization of BERT. In: AAAI (2019)
24. Sun, Z., Yu, H., Song, X., Liu, R., Yang, Y., Zhou, D.: MobileBERT: a compact task-agnostic BERT for resource-limited devices. In: ACL (2020)
25. Turc, I., Chang, M.W., Lee, K., Toutanova, K.: Well-read students learn better: on the importance of pre-training compact models. Preprint arXiv:1908.08962v2 (2019)
26. Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., Bowman, S.R.: GLUE: a multi-task benchmark and analysis platform for natural language understanding. In: ICLR (2019)
27. Wang, J., et al.: Revisiting parameter sharing for automatic neural channel number search. In: NeurIPS, vol. 33 (2020)
28. Wang, J., Bai, H., Wu, J., Cheng, J.: Bayesian automatic model compression. IEEE JSTSP **14**(4), 727–736 (2020)
29. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: HAQ: hardware-aware automated quantization with mixed precision. In: CVPR, pp. 8612–8620 (2019)
30. Wen, L., Zhang, X., Bai, H., Xu, Z.: Structured pruning of recurrent neural networks through neuron selection. NN **123**, 134–141 (2020)
31. Wu, J., et al.: PocketFlow: an automated framework for compressing and accelerating deep neural networks. In: NeurIPS, CDNNRIA workshop (2018)
32. Xu, J., et al.: Nas-BERT: task-agnostic and adaptive-size BERT compression with neural architecture search. In: KDD (2021)
33. Zhang, W., et al.: TernaryBERT: distillation-aware ultra-low bit BERT. In: EMNLP (2020)
34. Zhao, S., Gupta, R., Song, Y., Zhou, D.: Extremely small BERT models from mixed-vocabulary training. In: EACL (2021)
35. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: CVPR (2018)