TBERT: Dynamic BERT Inference with Top-k Based Predictors

Zejian Liu[1,2], Kun Zhao[1], Jian Cheng[1,2,3*]
[1]National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences
[2]School of Future Technology, University of Chinese Academy of Sciences
[3]AiRiA
{liuzejian2018, kun.zhao}@ia.ac.cn, jcheng@nlpr.ia.ac.cn

***Abstract*—Dynamic inference is a compression method that adaptively prunes unimportant components according to the input at the inference stage, which can achieve a better trade-off between computational complexity and model accuracy than static compression methods. However, there are two limitations in previous works. The first one is that they usually need to search the threshold on the evaluation dataset to achieve the target compression ratio, but the search process is non-trivial. The second one is that these methods are unstable. Their performance will be significantly degraded on some datasets, especially when the compression ratio is high. In this paper, we propose TBERT, a simple yet stable dynamic inference method. TBERT utilizes the top-k-based pruning strategy which allows accurate control of the compression ratio. To enable stable end-to-end training of the model, we carefully design the structure of the predictor. Moreover, we propose adding auxiliary classifiers to help the model's training. Experimental results on the GLUE benchmark demonstrate that our method achieves higher performance than previous state-of-the-art methods.**

***Index Terms*—Transformer, Dynamic Inference, Pruning**

I. Introduction

In the past few years, large-scale transformer-based [1] pretrained models have achieved great success in many fields, such as BERT [2] for natural language understanding, DERT [3] for object detection, and ViT [4] for image classification. However, it is difficult to deploy these models to resource-constrained devices (e.g., mobile phones and smart watches) due to the tremendous computation overhead and memory footprint. As a result, many compression methods have been proposed to accelerate inference, including quantization [5], static pruning [6], distillation [7], and dynamic inference (or dynamic pruning) [8]–[10].

Dynamic inference is gaining increased attention due to its flexibility and efficiency. Unlike static inference having a constant number of computations for each input, dynamic inference allows adaptively skipping the unimportant computations for a specific input instance with an appropriate skipping criterion. Intuitively, dynamic inference samples different sub-networks for different input instances at run-time to reduce the actual amount of calculation. Recent research efforts have demonstrated that dynamic inference can achieve a better trade-off between computational complexity and model accuracy.

While dynamic inference is promising, there are still some limitations in previous works. Firstly, these works are usually required to manually adjust a threshold to achieve the target compression ratio. Sometimes it is difficult to find a suitable threshold, since the compression ratio can be very sensitive to the threshold. For example, DeeBERT [9] introduces the entropy of the output probability as the early-exit criterion. The inference will be early stopped if the entropy is less than a preset threshold S. For RTE dataset, DeeBERT achieves a compression ratio of 20.4% when S is 0.6931. However, the compression ratio becomes 8.3% when S slightly increases to 0.6932. Secondly, the performance of the existing methods is not stable. For example, when the compression ratio is 60%, the accuracy drop of EBERT [11] is negligible if the datasets are large (e.g., MNLI), but its accuracy drop is up to 21.8% on those small datasets (e.g., RTE). On the contrary, DeeBERT performs better than EBERT on the RTE but suffer from a more significant accuracy drop on the MNLI.

To address the aforementioned limitations, we propose TBERT, a simple yet stable dynamic inference method. Specifically, we add a predictor in each layer to estimate the importance of different components, and the model will only retain k components with the highest importance score to achieve the expected compression ratio. As the k is computed according to the target compression ratio, there is no gap between the real and target compression ratios. However, the general design of predictor in previous dynamic methods [11]–[13] are not suitable for TBERT, which leads to an hard convergence of the model. To ensure the model can be trained end-to-end stably, We redesign the structure of the predictor from the perspective of optimization and model characteristics: 1) instead of using straight-through estimation (STE) [14] to solve the problem of non-differentiable, we use the sigmoid function to reduce the approximate gradient errors; 2) due to the computation graph of different samples in a batch is different, batch normalization (BN) [15] will collect wrong normalization statics. Therefore we use layer normalization (LN) [16] in the predictors. With the help of these two modifications, TBERT can be trained efficiently. Moreover, we propose adding auxiliary classifiers to help the model's training, which can further improve the performance, especially on those relatively small datasets. Experimental results demonstrate that TBERT achieves better performance than previous state-of-the-art methods on the eight tasks of the GLUE benchmark.

* Corresponding author.

978-3-9819263-7-8/DATE23/© 2023 EDAA

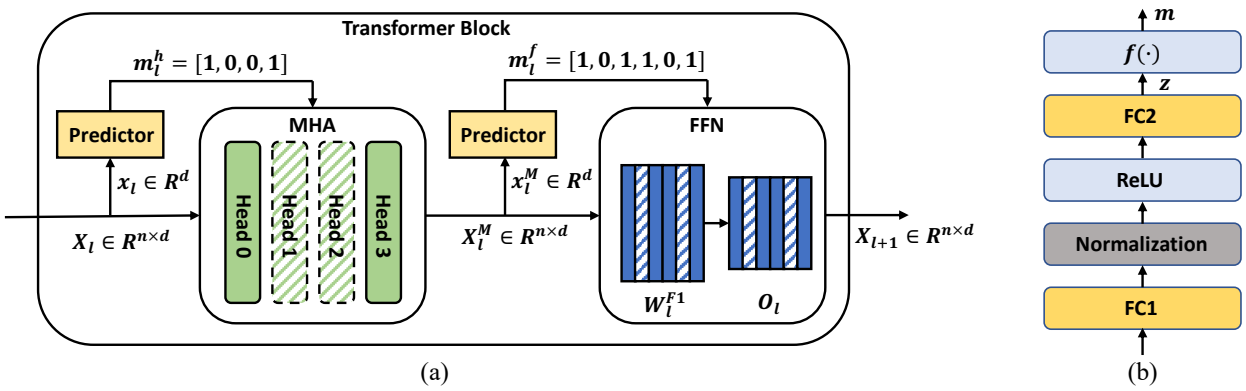


Fig. 1. (a) The overview of the dynamic pruning framework. It can be divided into BERT and predictors. We prune heads in MHA layers and intermediate dimensions in FFN layers; (b) The architecture of predictors.

II. Related Works

Dynamic inference (or pruning) is a compression method that can reduce the amount of actual calculation by removing different redundant computation components according to various inputs. Like static pruning methods, it can be divided into structured and unstructured inference.

A. Structured Dynamic Inference

Structured dynamic inference for Transformer models mainly tries to drop entire blocks, such as a layer or a head. These methods generally can be well supported by general-purpose hardware. However, the performance degrades a lot when the sparsity is too high. FastBERT [8], DeeBERT [9], and PABEE [10] dynamically adjust the number of layers to be executed. By adding additional classifiers in intermediate layers, the model can generate prediction results without passing through all layers. The main difference between these methods is the early-exit criterion. What's more, these works need to adjust a threshold to achieve different compression ratios. EBERT [11] dynamically skip unimportant heads in multi-head self-attention (MHA) and intermediate dimensions in point-wise feed-forward (FFN) with the help of predictors in each layer. It adds a sparsity loss function to ensure the trained model reaches the target compression ratio. Different from prior works, our method can accurately achieve the target compression ratio without adding a sparsity loss or adjusting a threshold.

B. Unstructured Dynamic Inference

Unstructured dynamic inference aims to remove individual activations or weights. For Transformer, these works mainly focus on the native sparsity of the attention matrices caused by the softmax operation. By designing effective prediction methods, they ignore those attention values that do not contribute to the final results to reduce the amount of computation. A^3 [17] presents an approximate candidate selection mechanism to remove the computation of unimportant attention scores. ELSA [18] proposes a hash-based method to find those pruned scores. Leopard [19] introduces a bit-level early-compute termination. The main problem with these methods is that special hardware is required to convert the theoretical acceleration ratio to the actual acceleration ratio.

III. Methods

A. Overall Framework

The overall architecture of the proposed method is shown in Figure 1(a). It consists of the original BERT and the extra predictors.

The BERT is mainly composed of L Transformer encoder blocks. Each encoder block consists of a MHA layer and a FFN layer. Suppose the length of an input sequence is n and the hidden-state size is d, the input and output of the l-th MHA layer that has N_h heads is:

$$MHA(X_l) = \sum_{i=1}^{N_h} m_{l,i}^h Att(X_l, W_{l,i}^Q, W_{l,i}^K, W_{l,i}^V, W_{l,i}^{O\top}), \quad (1)$$

where $W_{l,i}^Q, W_{l,i}^K, W_{l,i}^V, W_{l,i}^O \in R^{d \times d_h}$ are parameters matrices and $d_h = d/N_h$ denotes the output dimension of each head. $m_{l,i}^h \in \{0, 1\}$ is a mask variable that indicates whether the head has been pruned. For the original model, $m_{l,i}^h = 1$. Att$(\cdot)$ is the attention function.

Similarly, the input and output of the FFN layer that follows the l-th MHA layer is:

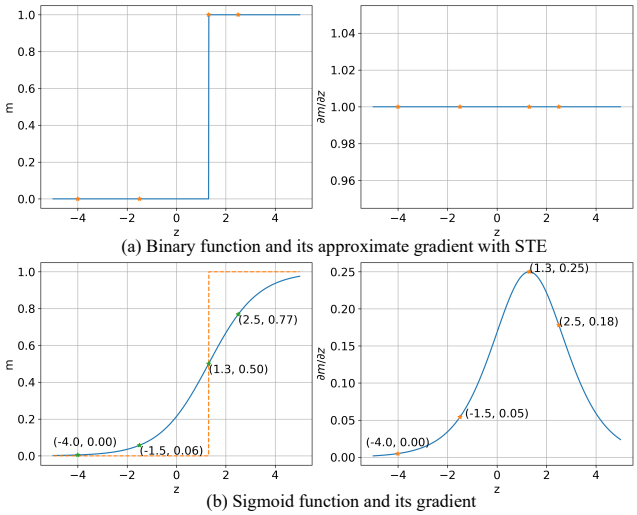
$$FFN(X_l^M) = GeLU(X_l^M W_l^{F1}) diag(m_l^f) W_l^{F2}, \quad (2)$$

where $W_l^{F1} \in R^{d \times d_I}$ and $W_l^{F2} \in R^{d_I \times d}$. $m_l^f \in \{0, 1\}$ also is a mask variable. If $m_{l,i}^f = 0$, $diag(m_l^f) \in R^{d_I \times d_I}$ will prune i-th intermediate dimension of $GeLU(X_l^M W_l^{F1})$.

Figure 1(b) presents the architecture of the predictor. It contains two FFN layers with a ReLU activation in between. The predictor determines the component to be pruned according to the current input instances. In other words, it generates m^h and m^f for each layer:

$$\mathbf{z} = ReLU(Norm(\mathbf{x} W_1^P)) W_2^P, \quad (3)$$

$$\mathbf{m} = f(\mathbf{z}), \quad (4)$$



(a) Binary function and its approximate gradient with STE

(b) Sigmoid function and its gradient

Fig. 2. The curve of functions and corresponding gradients. Assuming $\mathbf{z} = \{-4.0, -1.5, 1.3, 2.5\}$ and k=2, then $\mathbf{z}_{topk} = 1.3$. z_0, z_1 will be transformed to 0, and z_2, z_3 will be transformed to 1.

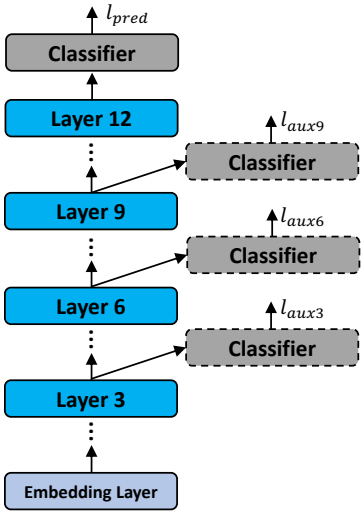


Fig. 3. Adding internal classifiers after several intermediate layers. The input of each internal classifier is hidden states of the connected layer, and the output is used to generate auxiliary losses. The dotted line represents these internal classifiers share the same architecture and parameters with the final classifier.

where Norm$(\cdot)$ is a normalization layer, and $f(\cdot)$ is a function that transform each z to a 0-1 mask. $\mathbf{x} \in R^d$ is features of the [CLS] token. This choice is based on the assumption that [CLS] representation encodes most of the useful information of the sentence [11].

In order to make the model reach the target compression ratio C (C = FLOPs of the compressed model / FLOPs of the original model) without adding any extra loss or designing a skip-criterion, a simple way is to use the top-k function as:

$$\mathbf{m} = f(\mathbf{z}) = \begin{cases} 1, & if \quad \mathbf{z} >= \mathbf{z}_{topk}, \\ 0, & otherwise. \end{cases} \tag{5}$$

where $\mathbf{z}_{topk}$ is the k-th largest value, and the k is computed according to the target compression ratio.

B. Reduce Approximate Gradient Errors

In order to train the entire model described above, we need to solve the problem that $f(\cdot)$ is not differentiable. A common practice is to use STE to compute the approximate gradient:

$$\frac{\partial m}{\partial z} = 1. \tag{6}$$

However, this approximation method introduces a significant gradient error because it does not consider that the contribution of different inputs to the gradients is different. As a result, the training process of the model becomes very unstable, which is shown in Figure 5. Similar phenomena also exist in quantization works [20].

Adding a differential function that approximates $f(\cdot)$ can reduce the gradient error, because it makes the backward propagation more consistent with the forward pass. We choose to use the sigmoid function due to the output of $f(\cdot)$ is 0 or 1:

$$\mathbf{z}' = \sigma(\mathbf{z} - \mathbf{z}_{topk}), \tag{7}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. The input is $\mathbf{z} - \mathbf{z}_{topk}$ because the value needs to compare with $\mathbf{z}_{topk}$, not zero. As $\mathbf{z}'$ is continuous, we need to transform it to 0-1 mask:

$$\mathbf{m} = g(\mathbf{z}') = \begin{cases} 1, & if \quad \mathbf{z}' >= 0.5, \\ 0, & otherwise. \end{cases} \tag{8}$$

Now we can obtain the approximate gradient of each z following (9). Figure 2 presents the gradient curve under different approximation methods.

$$\frac{\partial m}{\partial z} = \frac{\partial m}{\partial z'}\frac{\partial z'}{\partial z} = z'(1 - z'). \tag{9}$$

C. Normalization

Batch Normalization is widely used in deep neural networks, especially in computer vision (CV) tasks. It is commonly believed that BN solves the internal covariate shift issue [21] and improves the training efficiency by stabilizing activations with channel-wise mean and variance statistics estimated from samples in a mini-batch. Most of the previous predictor-based dynamic models for CV tasks add BN layers in their predictors [12], [13] to achieve stable training. However, as the empirical observation of previous works that BN is unsuitable for natural language processing (NLP) tasks, which leads to a significant performance drop, we found that using BN layers in predictors also makes the dynamic NLP model's training unstable, thus damaging the performance. Instead, layer normalization, which normalizes the inputs by computing the mean and variance of each feature and does not rely on the batch dimension, is helpful for the model's training. The detailed discussion is provided in Section IV-C.

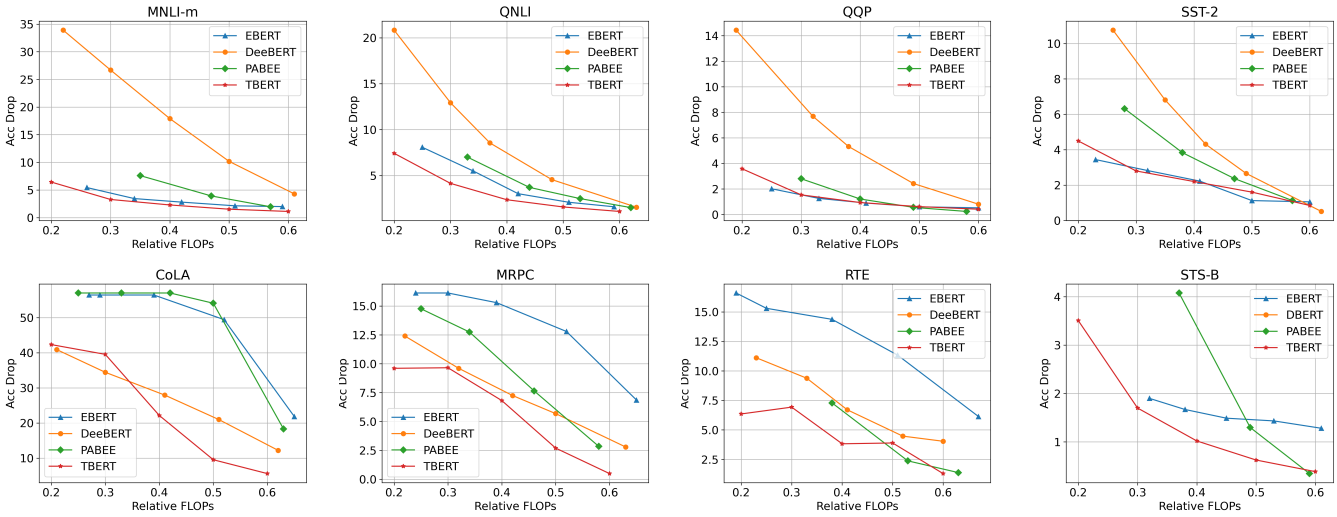


Fig. 4. Results on the development set of GLUE benchmark. The performance of DeeBERT on the STS-B dataset is not shown here because we do not find a threshold that can make the model reach the target compression ratio. We ignore the FLOPs of predictors due to the proportion is less than 0.1%.

D. Training with Auxiliary Classifiers

The well-trained predictors can predict the importance of different components for various input instances, and the quality of importance prediction significantly impacts the model's performance. Although we do not have any labels to supervise train the predictors, we can end-to-end train the top-k-based dynamic inference model with the help of sigmoid that reduces the approximate gradient error and LN that stable the training process. However, different from the BERT that has been pretrained on large datasets, the predictors are trained from scratch, so it requires more training steps to converge. When the training dataset is small, the number of training steps is often insufficient for the predictors to be fully trained. To solve this problem without increasing the number of training steps, we propose to add auxiliary losses, which is similar to inception networks [22]. Specifically, in addition to computing loss in the last classification layer, we add internal classifiers after several intermediate layers to get auxiliary loss, as shown in Figure 3. These internal classifiers share the same architecture and parameters as the last classifier to reduce the amount of trained parameters. The final loss is:

$$l = l_{pred} + \sum_{i \in I} l_{aux_i} \tag{10}$$

where I is a set of the index of internal layers that add auxiliary classifiers. Note that these auxiliary classifiers will be removed at the inference stage.

IV. Experiments

A. Experimental Setup

a) Datasets and Metrics.: To verify the effectiveness of our method, we conduct comprehensive experiments on eight GLUE tasks [23]: Multi-Genre Natural Language Inference Matched/Mismatched (MNLI-m/mm), Quora Question Pairs (QQP), Question Natural Language Inference (QNLI), Stanford Sentiment Treebank (SST-2), Recognizing Textual Entailment (RTE), Corpus of Linguistic Acceptability (CoLA), Microsoft Research Paraphrase Matching (MRPC), and Semantic Textual Similarity Benchmark (STS-B). We use accuracy as metrics for all tasks, except for Spearman Correlation for STS-B, and Matthews Correlation for CoLA.

b) Implementation details.: We implement TBERT with the HuggingFace Transformers Library v3.3.1 [24] and Pytorch v1.4.0 [25]. The model is BERT-base, and the hidden-state size of predictors is 64, which is the same as [11]. The training process is divided into two stages. In the first stage, we finetune the original model from the pretrained checkpoints downloaded from the HuggingFace Transformers. The hyperparameters, such as learning rate, epochs, and batch size are kept unchanged from the library for all downstream tasks. In the second stage, we add predictors in each layer and train the entire model. The initial learning rate for predictors is 1e-3, and we use a cosine learning scheduler to adjust the predictor's learning rate. Other hyperparameters are still unchanged.

Because the performance on small datasets (RTE, CoLA, STS-B, and MRPC) usually has large variance, we report the average performance over five runs with different random seeds for all experiments. All experiments are completed on a single Nvidia GeForce RTX2080Ti GPU.

B. Comparison with the Prior Methods

We compare our method with three dynamic inference methods designed for BERT, including DeeBERT [9], PABEE [10], and EBERT [11]. We reproduce these works using their released code, and the results reported are also the average of five runs. Because the baseline accuracy (accuracy of the model without pruning any components) of different methods has slight differences, we mainly report the accuracy drop instead of absolute accuracy.

Figure 4 shows the comparison of performance. We can see that other methods can not accurately achieve the target

TABLE I

ABLATION STUDY OF OUR PROPOSED THREE TECHNOLOGIES. ACT: WHETHER TO USE THE SIGMOID FUNCTION; NORM: THE TYPE OF NORMALIZATION LAYER IN THE PREDICTOR; AUX: WHETHER TO USE AUXILIARY CLASSIFIERS. THE RATIO OF RELATIVE FLOPS IS 20%.

Act.	Norm.	Aux.	MNLI	SST-2	RTE	MRPC
None	None	No	44.24	77.41	57.76	68.38
None	BN	No	34.74	79.7	54.87	68.38
None	LN	No	60.59	82.57	56.32	69.36
Sigmoid	None	No	66.85	86.93	59.21	76.72
Sigmoid	BN	No	69.86	84.06	55.96	68.38
Sigmoid	LN	No	77.76	88.07	61.01	77.21
Sigmoid	LN	Yes	**78.09**	**88.42**	**62.45**	**77.94**

compression ratio. For example, it is impractical for PABEE to achieve a 20% relative FLOPs ratio due to the limitation of its early-exit strategy. On the contrary, our method can easily achieve any given compression ratio with the help of the top-k-based pruning scheme. In terms of accuracy, our method consistently delivers similar or higher performance than baseline methods at each ratio of remaining FLOPs, especially on small datasets.

Specifically, for large datasets (MNLI, QQP, QNLI, and SST-2), the performance of EBERT is much better than DeeBERT and PABEE, and our method achieves comparable results with EBERT. However, EBERT's performance on small datasets is very poor. The main reason is that the number of training steps is insufficient to train predictors from scratch. DeeBERT performs well on CoLA, MRPC, and RTE, but its performance on STS-B is so bad that we can't draw its FLOPs-performance curve on the figure. Our method achieves a large performance improvement on four small datasets. For example, it achieves a 10.25% higher accuracy than EBERT on the RTE task when remaining 20% FLOPs.

C. Diccussion

We conduct an ablation study to investigate the effect of our proposed methods on the model performance, and the results are shown in Table I. We also show the loss curve of the model during training under different settings in Figure 5. The observation is that the model has the best performance when sigmoid, LN, and auxiliary classifiers are used simultaneously, especially for small datasets.

a) Sigmoid: Using the sigmoid function to reduce approximate gradient errors is essential for the training of the model. From Figure 5 we can see that the model without using the sigmoid function can not converge, even if we have added BN or LN. It means that the model cannot find the correct optimization direction due to the large gradient errors. When the sigmoid function is used, the model can be trained to converge whether normalization layers in predictors are used or not. There is also an interesting phenomenon. When the sigmoid function is used, z will gradually increase with the increase of training steps, as shown in Figure 6. It results in that $\sigma(z - z_{topk})$ is gradually closer to 0 or 1, which further reducing the approximat errors.

b) Normalization: The normalization layer is crucial for the stable training of models, but different models appreciate different types of normalization layers. Previous dynamic models generally use BN layers in predictors. However, the experimental results show that BN is unsuitable for our models. Sometimes it even causes performance degradation. For example, the performance of dynamic BERT without using BN and LN on SST-2 is 86.93%, which is higher than that of using BN. The main reason is that BN needs to compute the channel-wise mean and variance across the batch dimension. However, the dynamic inference model will prune different components for various inputs, which makes the BN collect inaccurate batch normalization statistics. This observation also exists in [26]. On the contrary, LN does not rely on the batch dimension to compute mean and variance, so it can correctly normalize the input.

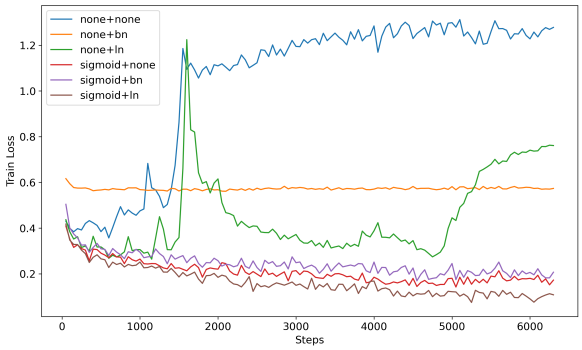


Fig. 5. Convergence curves of BERT with different settings on the SST-2 dataset. Lower is better. The ratio of relative FLOPs is 20%.

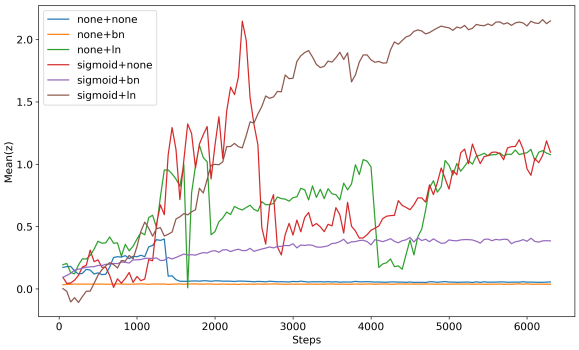
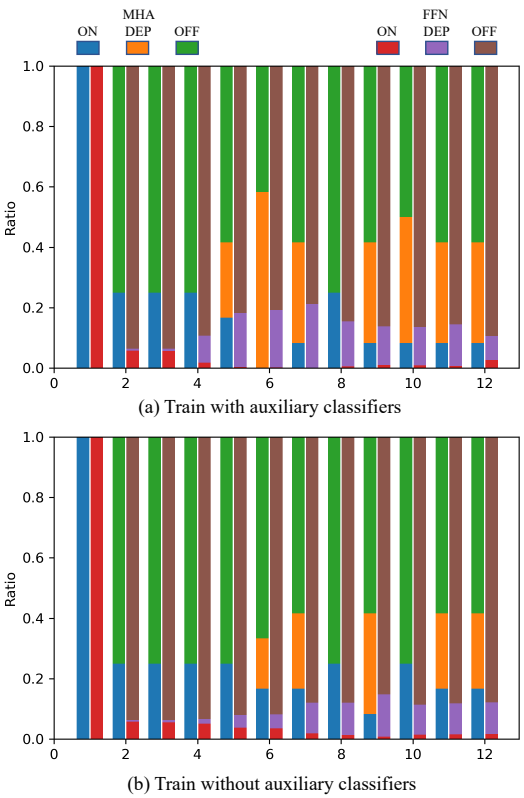


Fig. 6. The curve of training steps and the average value of $\mathbf{z}$. The model is trained on the SST-2 dataset and the ratio of relative FLOPs is 20%.

c) Auxiliary Classifiers: Table I shows that adding auxiliary classifiers can improve performance, especially for relatively small datasets. The main reason for the improvement of performance is that the intermediate classifiers can help the training process of the predictor so that it can more accurately predict the importance of different components. To demonstrate this, we visualize the mask distribution of different layers in Figure 7, like in [11], [27]. The mask m for each head or intermediate dimension can be divided into three classes: 1) on: be one for all inputs; 2) off: be zero for all inputs; 3) dep: be one or zero for different inputs. The higher the proportion of masks that belongs to the dep class, the stronger the dynamic



(a) Train with auxiliary classifiers

(b) Train without auxiliary classifiers

Fig. 7. Influence of auxiliary classifiers on dynamic characteristics. The task is RTE, and the target compression ratio is 20%. We do not add predictors in the first layer, so all masks belong to the on class.

adjustment ability of the model. We can see that model trained with auxiliary classifiers has a higher proportion of the dep class masks.

Similar to the view in [14], auxiliary classifiers are helpful for the model's training for two reasons. First, the existence of these intermediate predictors shortens the path of back propagation, so the gradients are increased. The second is these classifiers strengthen the discrimination ability of the intermediate layers. In other words, to maintain the discrimination ability, the predictors must accurately find those unimportant components to reduce the loss of information. It is equivalent to providing some supervision information for the training of the predictors, which accelerates the model's convergence.

V. Conclusion

In this paper, we propose TBERT, a simple but effective and stable dynamic inference method. Specifically, to achieve the target compression ratio accurately without tuning any threshold or hyperparameters, we design a top-k-based predictor. Compared with the predictors that previous works used, the key modifications are using the sigmoid function instead of STE and LN instead BN. The model that utilizes the newly designed predictors can be trained end-to-end. Moreover, we propose adding auxiliary classifiers to help the training further. Experimental results demonstrate that our method achieves a better performance on all tasks of the GLUE benchmark than previous state-of-the-art methods with similar compression ratios.

VI. Acknowledgment

This work was supported in part by the National Key Research and Development Program of China (Grant No. 2021ZD0201504), National Natural Science Foundation of China (No.61972396, 61906193), the Strategic Priority Research Program of Chinese Academy of Sciences (No. XDB32050200), Jiangsu Key Research and Development Plan (No.BE2021012-2).

References

[1] A. Vaswani *et al.*, "Attention is all you need," in *NeurIPS*, p. 6000–6010, 2017.
[2] J. Devlin *et al.*, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, pp. 4171–4186, June 2019.
[3] N. Carion *et al.*, "End-to-end object detection with transformers," in *ECCV*, pp. 213–229, 2020.
[4] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2021.
[5] O. Zafrir *et al.*, "Q8bert: Quantized 8bit bert," in *NeurIPS workshop*, pp. 36–39, IEEE, 2019.
[6] P. Michel *et al.*, "Are sixteen heads really better than one?," in *NeurIPS*, vol. 32, pp. 14014–14024, 2019.
[7] C. Xu *et al.*, "BERT-of-theseus: Compressing BERT by progressive module replacing," in *EMNLP*, pp. 7859–7869, Nov. 2020.
[8] W. Liu *et al.*, "FastBERT: a self-distilling BERT with adaptive inference time," in *ACL*, pp. 6035–6044, July 2020.
[9] J. Xin *et al.*, "DeeBERT: Dynamic early exiting for accelerating BERT inference," in *ACL*, pp. 2246–2251, July 2020.
[10] W. Zhou *et al.*, "Bert loses patience: Fast and robust inference with early exit," in *NeurIPS*, 2020.
[11] Z. Liu *et al.*, "EBERT: Efficient BERT inference with dynamic structured pruning," in *Findings of ACL*, pp. 4814–4823, Aug. 2021.
[12] B. E. Bejnordi *et al.*, "Batch-shaping for learning conditional channel gated networks," in *ICLR*, 2020.
[13] F. Li *et al.*, "Dynamic dual gating neural networks," in *ICCV*, pp. 5310–5319, 2021.
[14] Y. Bengio *et al.*, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.
[15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, pp. 448–456, PMLR, 2015.
[16] J. L. Ba *et al.*, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.
[17] T. J. Ham *et al.*, "A3: Accelerating attention mechanisms in neural networks with approximation," in *HPCA*, pp. 328–341, 2020.
[18] T. J. Ham *et al.*, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *ISCA*, pp. 692–705, 2021.
[19] Z. Li *et al.*, "Accelerating attention through gradient-based learned runtime pruning," in *ISCA*, p. 902–915, 2022.
[20] R. Gong *et al.*, "Differentiable soft quantization: Bridging full-precision and low-bit neural networks," in *ICCV*, pp. 4852–4861, 2019.
[21] H. Shimodaira, "Improving predictive inference under covariate shift by weighting the log-likelihood function," *Journal of Statistical Planning and Inference*, vol. 90, no. 2, pp. 227–244, 2000.
[22] C. Szegedy *et al.*, "Going deeper with convolutions," in *CVPR*, pp. 1–9, 2015.
[23] A. Wang *et al.*, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *EMNLP*, pp. 353–355, Nov. 2018.
[24] T. Wolf *et al.*, "Transformers: State-of-the-art natural language processing," in *EMNLP*, pp. 38–45, Oct. 2020.
[25] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, vol. 32, 2019.
[26] J. Yu *et al.*, "Slimmable neural networks," in *ICLR*, 2019.
[27] Z. Chen *et al.*, "You look twice: Gaternet for dynamic filter selection in cnns," in *CVPR*, June 2019.