GAML-BERT: Improving BERT Early Exiting by Gradient Aligned Mutual Learning

Wei Zhu[1,2], **Xiaoling Wang**[1]*, **Yuan Ni**[2], **Guotong Xie**[2,3,4]
Zhen Guo[5], **Xiaoming Wu**[5]
[1] East China Normal University, Shanghai, China
[2] Ping An Health Tech, Beijing/Shanghai, China
[3] Ping An Healthcare and Tech, Shanghai, China
[4] Ping An International Smart City, Shenzhen, China
[5] Qingdao Eye Hospital of Shandong First Medical University, Qingdao, China

Abstract

In this work, we propose a novel framework, Gradient Aligned Mutual Learning BERT (GAML-BERT), for improving the early exiting of BERT. GAML-BERT's contributions are two-fold. We conduct a set of pilot experiments, which shows that mutual knowledge distillation between a shallow exit and a deep exit leads to better performances for both. From this observation, we use mutual learning to improve BERT's early exiting performances, that is, we ask each exit of a multi-exit BERT to distill knowledge from each other. Second, we propose GA, a novel training method that aligns the gradients from knowledge distillation to cross-entropy losses. Extensive experiments are conducted on the GLUE benchmark, which shows that our GAML-BERT can significantly outperform the state-of-the-art (SOTA) BERT early exiting methods.

1 Introduction

Since BERT (Devlin et al., 2018), the pre-trained language models (PLMs) are dominating the field of natural language processing (NLP). The recent years have witnessed the rise of many PLMs, such as GPT (Radford et al., 2019), XLNet (Yang et al., 2019), and ALBERT (Lan et al., 2020), and so forth. These BERT-style models achieved considerable improvements in many Natural Language Processing (NLP) tasks by pre-training on the unlabeled corpus and fine-tuning on labeled tasks, such as text classification natural language inference (NLI), sequence labeling. However, PLMs are notorious for being gigantic and slow in both training and inference. Their significant inference latencies pose great challenges to deployment in real-time applications, such as chat-bots and search engines.

In addition, previous literature (Fan et al., 2020; Michel et al., 2019; Zhou et al., 2020) find that large PLMs with dozens of Transformer layers are over-parameterized and suffer from the "overthinking" problem (Kaya et al., 2019). That is, for many input samples, their shallow representations at a shallow layer are enough to make a correct classification. Moreover, the final layer's representations may be too overfitting to generalize.[1] The overthinking problem leads to not only poor generalization but also wasted computation.

Addressing the above two issues, a branch of literature focuses on making PLMs' inference more efficient via network pruning (Zhu and Gupta, 2018; Xu et al., 2020; Fan et al., 2020; Michel et al., 2019; Zhu et al., 2021; Zhu, 2021c,a), knowledge distillation (Sun et al., 2019b; Sanh et al., 2019; Jiao et al., 2020), weight quantization (Zhang et al., 2020; Bai et al., 2020; Kim et al., 2021) and adaptive inference (Zhou et al., 2020; Xin et al., 2020; Liu et al., 2020). The adaptive inference has drawn much attention. The adaptive inference is designated to process simple examples with only shallow layers of BERT and predict more difficult queries with deeper layers, thus significantly speeding up the inference time on average while maintaining high accuracy. The speed-up ratio can be controlled with certain hyper-parameters to handle drastic changes in request traffic. What is more, it can address the over-thinking problem and improve the model's generalization ability.

Early exiting is one of the most important adaptive inference methods (Bolukbasi et al., 2017). As depicted in Figure 1, it implements adaptive inference by installing an early exit, i.e., an intermediate prediction layer, at each layer of BERT and early exiting "easy" samples to speed up inference. At the training stage, all the exits are jointly optimized with BERT's parameters. At the inference stage, there are two different settings. First, in budgeted exiting mode, the model makes a prediction with

*Corresponding author. Email: xlwang@cs.ecnu.edu.cn.

[1]The over-thinking problem is observed on SST-2 and CoLA. The details can be found in the Appendix.

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 3033–3044
November 7–11, 2021.

a fixed exit for all queries. This mode deals with heavy traffic by assigning a shallower exit for prediction. The other one is dynamic exiting mode. That is, some strategies for early exiting is designed to decide whether to exit at each layer given the currently obtained predictions (from previous and current layers) (Teerapittayanon et al., 2016; Kaya et al., 2019; Xin et al., 2020; Zhou et al., 2020). In this mode, different samples can exit at different depths.

Knowledge distillation (KD) (Hinton et al., 2015) is of essential importance for improving early exiting performances. The traditional belief of KD is that the teacher model (usually a stronger model) teaches a lower-capacity student model "dark knowledge" through providing soft targets. As an application of this traditional belief, recent studies by Phuong and Lampert (2019); Liu et al. (2020) improve the training procedure by incorporate KD losses, which encourages the early exits to mimic the output distributions of the last exit. Yuan et al. (2019) challenge the common belief of KD by revealing that knowledge distillation is actually a learned label smoothing (LS) regularization (Szegedy et al., 2016), and a weaker teacher can also enhance a stronger student's performance via knowledge distillation. Zhu (2021b) shows that asking all the exits to learn from one another (mutual learning) is beneficial for early exiting. Sun et al. (2019a) introduce dense pairwise knowledge matching operations at certain intermediate layers of deep convolutional networks during training, which are demonstrated to be beneficial for the whole network's generalization. However, Zhu (2021b) propose this mutual learning framework intuitively, and does not fully explore the underlying motivations. Sun et al. (2019a) focuses on improving the whole network, and does not investigate how these mutual learning (or knowledge matching) affects each layer's exiting performances.

In this work, we first conduct a series of exploratory experiments called pairwise mutual learning (PML). PML selects two exits of BERT and consider the finetuning of these exits with or without adding different knowledge distillation settings. Our experiments show the following three approaches can imrpove the shallower exit's performances: (1) adding supervisions to deeper exits; (2) KD from deeper exits; (3) further asking the deeper exits to learn from this shallow exits. In addition, via the above approaches, the deeper exit's performance also improves. Our experimental findings is consistent with and complement the conclusions of Szegedy et al. (2016) and Sun et al. (2019a). And thus we adopt the mutual learning (ML) framework to enhance the training of early exits. ML asks all the exits to learn from one another (depicted by Figure 1), thus fully releasing knowledge transfer and regularization capabilities of KD.

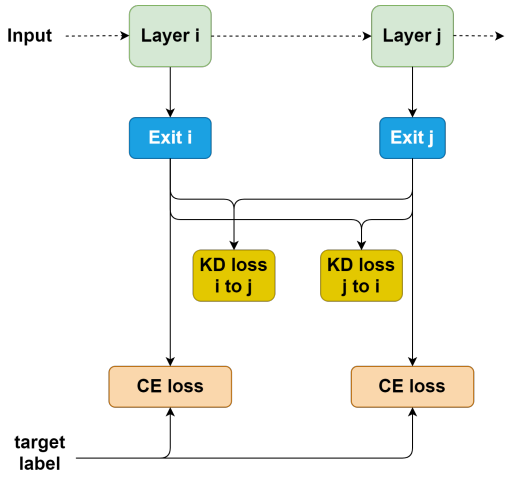
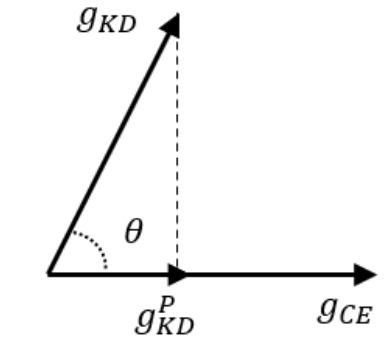


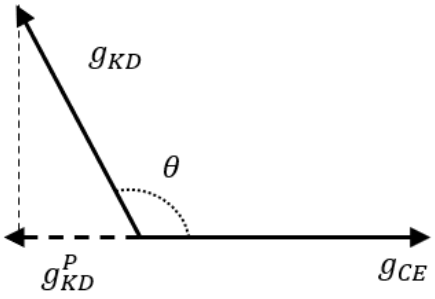
Figure 1: An illustration of the mutual learning framework.

Further, by analyzing the directions of gradients from cross-entropy loss and distillation loss (denoted as $\mathbf{g}_{CE}$ and $\mathbf{g}_{KD}$), we find that $\mathbf{g}_{KD}$ is often in a conflicting direction with $\mathbf{g}_{CE}$. We hypothesize that $\mathbf{g}_{KD\perp CE}$, $\mathbf{g}_{KD}$'s part that is orthogonal with $\mathbf{g}_{CE}$, will extract the model from moving toward optimum. Thus we propose a novel optimization mechanism called Gradient Alignment (GA). As depicted by 2, GA will project $\mathbf{g}_{KD}$ onto the direction of $\mathbf{g}_{CE}$. We propose two versions of GA, GA-soft, and GA-hard, which only differ in how to deal with $\mathbf{g}_{KD}$ when the angle of the two gradients is larger than 90°.

We will call our framework Gradient Aligned Mutual Learning for BERT (GAML-BERT). Extensive experiments are conducted on the GLUE benchmark (Wang et al., 2018) and show that GAML-BERT outperforms existing SOTA BERT early exiting methods, sometimes by a large margin. Deeper analysis and ablation studies result in the following main takeaways: (a) knowledge distillation among the exits can improve their performances, especially for the shallow ones; (b) our gradient alignment method can improve the training procedure and thus improve the model's generalization performances.



(a) angle <= 90°



(b) angle > 90°

Figure 2: Two scenarios of our proposed gradient alignment method. Note that when the gradients' angle is larger than 90°, $\mathbf{g}^P_{KD}$ is opposite to $\mathbf{g}_{CE}$.

Our contributions are summarized as follows:

- We conduct exploratory experiments to demonstrate that the mutual knowledge distillation between two exits of different depth are benefical for both .
- We propose a novel gradient alignment method for better optimization.

2 Preliminaries

In this section, we introduce the necessary background for BERT early exiting. Throughout this work, we consider the case of multi-class classification with samples $\{(x, y), x \in \mathcal{X}, y \in \mathcal{Y}, i = 1, 2, ..., N\}$, e.g., sentences, and the number of classes is K.

2.1 Backbone models

In this work, we adopt BERT as backbone models. BERT is a multi-layer Transformer (Vaswani et al., 2017) network, which is pre-trained in a self-supervised manner on a large corpus.

2.2 Early-exiting Architecture

As depicted in Figure 1, early exiting architectures are networks with exits[2] at each transformer layer. With M exits, M classifiers $f_m(x; \theta_m) : \mathcal{X} \rightarrow \Delta^K$ $(m = 1, 2, ..., M)$ are designated at M layers of BERT, each of which maps its input to the probability simplex Δ^K, i.e., the set of probability distributions over the K classes. All the parameters of the transformer layers and exits are denoted as Θ.

2.2.1 Training

At the training stage, all the exits are jointly optimized with a summed loss function. Following Huang et al. (2017) and Zhou et al. (2020), the loss function is the weighted average of the cross-entropy (CE) losses given by

$$\mathcal{L}^{CE} = \frac{\sum_{m=1}^{M} m * \mathcal{L}_m^{CE}}{\sum_{m=1}^{M} m}, \quad (1)$$

where $\mathcal{L}_m^{CE} = \mathcal{L}_m^{CE}(y, f_m(x; \theta_m))$ denotes the cross-entropy loss of the m-th exit. Note that the weight m corresponds to the relative inference cost of exit m.

2.2.2 Inference

At inference, the multi-exit BERT can operate in two different modes, depending on whether the computational budget to classify an example is known or not.

Budgeted Exiting. If the computational budget is known, we can directly appoint a suitable exit of BERT, $f_{M'}(x; \theta_{M'})$, to predict all queries.

Dynamic Exiting. Under this mode, after receiving a query input x, the model starts to predict on the classifiers $f_1(x; \theta_1)$, $f_2(x; \theta_2)$, ..., in turn in a forward pass, reusing computation where possible. It will continue to do so until it receives a signal to stop early at an exit $M'' < M$, or arrives at the last exit M. At this point, it will output the final predictions based on the current and previous predictions. Note that under this early exit setting, different samples might exit at different layers. [3]

3 Pilot Experiment and Analysis

3.1 Pilot experiments

In this section, we examine the effects of mutual learning among exits by conducting a series of

[2]Some literature (e.g., DeeBERT (Xin et al., 2020)) also refers to exits as off-ramps.

[3]We provide a short review of the dynamic exiting strategies in the Appendix for further reference.

pilot experiments called pairwise mutual learning (PML). In the PML experiments, we select two exits (i, j) ($i < j$, i.e., exit i is shallower than exit j). We consider the following settings:

- Directly finetuning exit x ($x = i, j$). In this setting, we reveal supervision signals to exit x and finetune it among with the BERT parameters.
- Finetuning exit i and j jointly. That is, we sum up the losses of exit i and j during training.
- Finetuning exit i and j jointly, and asking exit i to learn from exit j.
- Finetuning exit i and j jointly, and asking exit j to learn from exit i.
- Finetuning exit i and j jointly, and asking the two exits to learn from each other.

We conduct the above PML experiments on CoLA and SST-2 datasets in the GLUE benchmark (Wang et al., 2018). We select two exit pairs, (1, 12) and (6, 12). The performance metrics follow GLUE. Detailed experimental settings are reported in the Appendix.

3.2 Result analysis

Table 1 reports the results of our pilot experiments. From the results we can see that:

- Exit 12 benefis from KD from exit 6, demonstrating that the last exit, as a strong student, still obtain performance improvements when it receives knowledge distillation from a much weaker teacher. This observation is consistent with Yuan et al. (2019).
- When a lower exit (1 or 6) is finetuned jointly with exit 12 (with no KD), both exits' performances will improve. This observation is consistent with Sun et al. (2019a). Intuitively, letting the intermediate layers to receive supervision signals can improve the lower layers' representation capabilities, thus helping the last exit. For lower layers, receiving the top layers' gradient signal is benefical for lower layers's optimization, thus the lower exit's performance can also be significantly boosted.
- It is normal for lower exits to improve significantly when it receives KD signals from exit 12 since it receives superior knowledge from the latter (Hinton et al., 2015). However, we can see that the lower exits' performances further improve when we introduce mutual KD between the exit pair. Mutual learning not only improve the last exit (consistent with Sun et al. (2019a)) but also the lower exits. We believe the low exits' extra performance gains are from: (a) a better top layer, thus gradient signals are better; (b) mutual learning drives the behaviors of the exit pair to be more similar, which is like a regularization that help to improve the generalization performances.

-	Task	SST-2	CoLA
BERT-base	-	91.5	54.7
Exit 1 and Exit 12			
Exit 1	directly finetuning	56.4	0.0
	finetune with exit 12	61.5	0.0
	KD from exit 12	74.6	0.0
	mutual KD with exit 12	75.8	0.0
Exit 12	directly finetuning	91.5	54.7
	finetune with exit 1	91.6	55.0
	KD from exit 1	91.3	55.1
	mutual KD with exit 1	91.5	55.3
Exit 6 and Exit 12			
Exit 6	directly finetuning	88.6	36.4
	finetune with exit 12	88.9	39.9
	KD from exit 12	89.3	46.5
	mutual KD with exit 12	89.6	47.6
Exit 12	directly finetuning	91.5	54.7
	finetune with exit 6	91.8	55.4
	KD from exit 6	92.0	55.9
	mutual KD with exit	92.2	56.7

Table 1: The results of our PML experiments. We report dev performances on SST-2 and CoLA.

4 Mutual learning

In light of the above analysis, and following Zhu (2021b) and Sun et al. (2019a), we adopt the mutual learning framework (Figure 1) to explore the potentials of early exits. That is, all the exiting classifiers learn from one another. The loss terms from this fully mutual learning framework are added to the cross-entropy losses in Eq. 1, and the loss objective becomes

$$\mathcal{L} = (1-\alpha)\mathcal{L}^{CE} + \alpha\mathcal{L}^{KD}, \tag{2}$$

where $\mathcal{L}^{KD}$ is given by

$$\mathcal{L}^{KD} = \sum_{i=1}^{M}\sum_{j\neq i}\mathcal{L}^{KD}_{i\rightarrow j}. \tag{3}$$

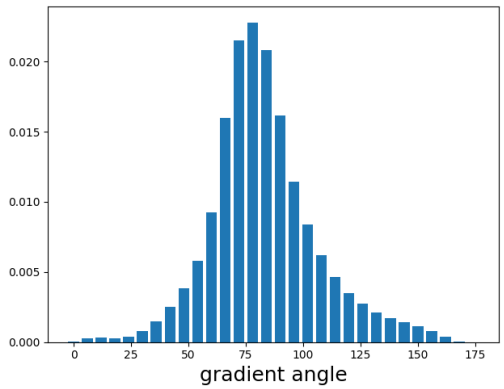


Figure 3: The distribution of the gradient angles when the BERT model is fine-tuned on SST-2 with the mutual learning training objective.

The ML framework is different from FastBERT (Liu et al., 2020) in two aspects. First, FastBERT employs a two-stage learning mechanism, where the optimization with KD is separated from optimization with the cross-entropy loss, and the BERT backbone is frozen during the optimization with KD losses. Second, FastBERT only asks the lower exits to distill knowledge from the last layer. In the ML framework, each layer receives the regularizations from all other exits, thus fully exploiting the regularization potentials of knowledge distillation. In this work, we run FastBERT with the codes of Liu et al. (2020), and experimental results will demonstrate that the ML framework outperforms FastBERT.

5 Gradient alignment

The ML training mechanism can stabilize the training process and lead to better optimization by implementing rich regularizations over all the exits. However, during experiments, we still observe that the weight of the KD loss α has a large impact on the model performances, and the better performances are achieved by setting α to be relatively small (e.g., 0.1 or 0.2). Our observations are consistent with the experimental observations of Yang et al. (2020); Sun et al. (2019b). Intuitively, it seems that the KD objective is conflicting with the CE loss to a certain degree.

To visualize the interactions between cross-entropy loss and KD loss, we separately compute the gradients derived from the two loss objectives, $\mathbf{g}_{KD} = \Delta_\Theta \mathcal{L}_{KD}$ and $\mathbf{g}_{CE} = \Delta_\Theta \mathcal{L}_{CE}$. Their angle is given by

$$\cos\gamma = \frac{\mathbf{g}_{KD} \cdot \mathbf{g}_{CE}}{|\mathbf{g}_{KD}| \cdot |\mathbf{g}_{CE}|}, \quad \gamma = Argcos(\cos\gamma). \tag{4}$$

During the BERT model fine-tuning on SST-2 with the mutual learning training objective, we calculate the angles of the gradients $\mathbf{g}_{KD}$ and $\mathbf{g}_{CE}$ on each training step. The distribution of the gradient angles are plotted in Figure 3. We can see that about half of the optimization steps γ is larger than 90°, meaning that they have conflicting directions for optimization. This observation motivates us to think that we may obtain better convergences if we can somehow align the two gradients. Thus, the above observation naturally leads to the following hypothesis:

Hypothesis 1 (H1): *Dropping off the part of KD's gradient that conflicts with CE's gradient and only keeps the part aligned with the latter can improve the trained model's performances.*

Thus, we propose a novel optimization method, gradient alignment (GA), to align KD's gradient $\mathbf{g}_{KD}$ with CE's gradient $\mathbf{g}_{CE}$. GA is depicted in Figure 2. When we project $\mathbf{g}_{KD}$ on $\mathbf{g}_{CE}$, the projected vector is given by

$$\mathbf{g}^P_{KD} = \frac{|\mathbf{g}_{KD}|\cos\theta}{|\mathbf{g}_{CE}|}\mathbf{g}_{CE}. \tag{5}$$

Denote the final modified gradient as $\mathbf{g}_{GA}$. We propose two versions of GA, as follows.

GA-soft When the angle θ is larger than 90°, $\mathbf{g}^P_{KD}$ is also added to $\mathbf{g}_{CE}$. Thus $\mathbf{g}_{GA}$ is given by

$$\mathbf{g}_{GA} = (1-\alpha)\mathbf{g}_{CE} + \alpha\mathbf{g}^P_{KD}. \tag{6}$$

In GA-soft, when the angle θ is larger than 90°, $\mathbf{g}^P_{KD}$ is in the opposite direction with $\mathbf{g}_{CE}$, thus might slow down or reverse this gradient descent direction.

GA-hard When the angle θ is larger than 90°, $\mathbf{g}^P_{KD}$ is not added to $\mathbf{g}_{CE}$. Thus $\mathbf{g}_{GA}$ is given by

$$\mathbf{g}_{GA} = \begin{cases} (1-\alpha)\mathbf{g}_{CE} + \alpha\mathbf{g}^P_{KD}, & \text{if } \theta \le 90° \\ \mathbf{g}_{CE}, & \text{otherwise} \end{cases} \tag{7}$$

In GA-hard, when the angle θ is larger than 90°, we discard $\mathbf{g}^P_{KD}$.

Our proposed method, GA, is intuitively sound. In GA, the gradient descent direction strictly follows $\mathbf{g}_{CE}$, and we discard the part of $\mathbf{g}_{KD}$ that is

orthogonal to $\mathbf{g}_{CE}$, thus eliminating the conflicting signals from $\mathbf{g}_{KD}$. What is more, the projected gradient $\mathbf{g}_{KD}^{P}$ can help to adjust the pace of gradient descent. When the angle θ is smaller than 90°, the projected gradient $\mathbf{g}_{KD}^{P}$ is added to $\mathbf{g}_{CE}$. In this scenario, the gradients have similar directions. Thus the optimizer is quite sure of the optimization direction, and it should move with a larger step. When the angle θ is larger than 90°, $\mathbf{g}_{KD}^{P}$ is in the opposite direction with $\mathbf{g}_{CE}$. On the one hand, $\mathbf{g}_{KD}^{P}$ can be seen as a regularization to $\mathbf{g}_{CE}$, and stops $\mathbf{g}_{CE}$ from local optimum or jumping away from optimum. On the other hand, $\mathbf{g}_{KD}^{P}$ might slow down the convergences. Thus, we will leave the selection between GA-soft and GA-hard as a hyper-parameter.

6 Experiments

6.1 Datasets

We evaluate our proposed approach to the classification tasks on the GLUE benchmark. We only exclude the STS-B task since it is a regression task, and we exclude the WNLI task following previous work (Devlin et al., 2018; Xu et al., 2020).

6.2 Backbone models

Backbone models. All of the experiments are built upon the Google BERT (Devlin et al., 2019). We ensure fair comparison by setting the hyper-parameters related to the PLM backbones the same as HuggingFace Transformers (Wolf et al., 2020).

6.3 Baseline methods

We compare with the previous BERT early exiting methods and compare other methods that speed up BERT inference.

Directly reducing layers. We experiment with directly utilizing the first 6 layers of the original (AL)BERT with a single output layer on the top, denoted by (AL)BERT-xL ($x = 6$). This baseline serves as a lower bound for performance matrics since it does not employ any additional technique.

Static model compression approaches. For knowledge distillation, we include DistillBERT (Sanh et al., 2019) and BERT-PKD (Sun et al., 2019b).[4] For model parameter pruning, we include the results of LayerDrop (Fan et al., 2020) and attention head pruning (Michel et al., 2019) on ALBERT. For module replacing, we include BERT-of-Theseus (Xu et al., 2020).

Early exiting approaches. We compare our method with the previous state-of-the-art BERT early exiting approaches, under both budgeted exiting mode and dynamic exiting mode. For dynamic exiting mode, we compare with: (a) entropy-based method DeeBERT; (b) score-based method Shallow-deep; and (c) patience-based exiting method PABEE; (d) FastBERT when it adopts the PABEE's exiting strategy. For budgeted exiting mode, we compare with: (a) BERT with Multi-exits fine-tuned with a loss objective given by Equation 1, which DeeBERT and PABEE adopt; (b) FastBERT.

6.4 Experimental settings

We implement our GAML-BERT and other baseline methods based on HuggingFace's Transformers (Wolf et al., 2020). We conduct our experiments on a single Nvidia V100 16GB GPU.

Training. We add a linear output layer after each intermediate layer of the pre-trained BERT/ALBERT model as the internal classifier. **The hyperparameter tuning is done in a cross-validation fashion on the training set so that the dev set of GLUE tasks remains blind for model generalization.** We perform grid search over batch sizes {16, 32, 128}, and learning rates {1e-5, 2e-5, 3e-5, 5e-5} for model parameters Θ, and warm-up steps of {0.8, 1.0, 1.2} times the number of steps in an epoch, and values of weight α (from Eq. 2) {0.1, 0.2, 0.3, 0.5, 0.8}. We will adopt the Adam optimizer. We apply an early stopping mechanism with patience 5 and evaluate the model on the valid set (from cross-validation) after each epoch. Moreover, we define the dev performance of our early exiting architecture as the average performance of all the exits. We will select the model checkpoint with the best average performance in cross-validation.

Dynamic exiting mode inference. Following prior work (Zhou et al., 2020), dynamic exiting mode inference is on a per-instance basis, i.e., the batch size for inference is set to 1. We believe this setting mimics the common latency-sensitive production scenario when processing individual requests of different difficulties from different users. We adjust the hyper-parameters for each dynamic exiting method such that the speed-up ratio is between 1.80x to 2.1x.

[4] Note that the two methods consider knowledge distillation on the fine-tuning stage, whereas TinyBERT (Jiao et al., 2020) and Turc et al. (2019) investigate knowledge distillation during both the pre-training stage and fine-tuning stage.

Method	#Param	Speed-up	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2
Dev set									
BERT-base	109M	1.00x	54.7	83.5	88.2	86.8	88.7	67.1	91.5
BERT-6L	66M	1.96x	36.4	77.1	83.9	80.2	84.8	60.3	88.6
DistillBERT	66M	1.96x	46.5	79.8	85.3	82.2	86.4	63.1	89.3
BERT-PKD	66M	1.96x	47.4	80.6	85.5	82.5	86.8	63.4	89.7
LayerDrop	66M	1.96x	43.2	80.1	85.2	81.7	86.2	62.8	89.1
BERT-of-Theseus	66M	1.96x	44.5	80.7	85.4	82.4	86.5	63.6	89.6
DeeBERT	109M	1.88x	43.4	81.2	85.8	82.5	87.3	63.9	90.0
Shallow-Deep	109M	1.95x	44.5	81.1	85.7	82.6	87.2	64.1	90.1
PABEE	109M	1.91x	45.2	81.5	86.2	83.1	87.5	64.5	90.5
FastBERT	109M	1.93x	48.6	82.1	86.7	83.6	88.1	64.9	90.8
FastBERT-GA (ours)	109M	1.95x	50.4	82.6	87.5	84.7	88.5	66.2	91.4
ML-BERT (ours)	109M	1.96x	49.2	82.7	87.3	84.2	88.6	65.7	91.3
GAML-BERT (ours) (GA-hard)	109M	1.95x	50.5	83.1	87.6	84.9	88.9	66.3	91.5
GAML-BERT (ours) (GA-soft)	109M	1.95x	**51.1***	**83.2***	**88.1***	**85.1***	**89.1***	**66.8***	**91.9***
Test set									
BERT-base	109M	1.00x	50.6	83.4	87.2	85.7	70.4	64.7	92.7
PABEE	109M	1.89x	44.8	81.6	85.2	82.2	69.2	62.1	91.3
FastBERT	109M	1.95x	45.7	82.0	85.7	82.6	69.8	62.6	91.7
GAML-BERT (ours) (GA-soft)	109M	1.96x	**47.2***	**83.3***	**87.2***	**84.2***	**70.9***	**64.3***	**92.8***

Table 2: Experimental results of models with BERT backbone on the GLUE's development set and test set. This table reports the results under the dynamic exiting mode inference. The mean performance scores of 5 runs are reported. The speed-up ratio is averaged across seven tasks. Best performances are bolded, "*" indicates the performance gains by our full model GAML-BERT against the baseline models by the literature are statistically significant.

Budgeted exiting mode inference. In this setting, a multi-exit model is forced to output its prediction with a given exit. The results under this mode will be mainly reported in figures depicting the relation between the depth of the exit and the performance scores.

6.5 Overall Comparison

Table 2 reports the main results on GLUE with BERT as the backbone model under the dynamic exiting inference mode. From Table 2, we can see that our full model GAML-BERTs, especially with GA-soft, outperforms all previous methods to improve inference efficiency while maintaining good performances, demonstrating the proposed GAML-BERT framework's effectiveness. Note that Table 2 shows that GAML-BERT with GA-soft outperforms GA-hard consistently and by a clear margin on CoLA, MRPC, RTE. Thus, we will refer to GAML-BERT with GA-soft as our GAML-BERT model.

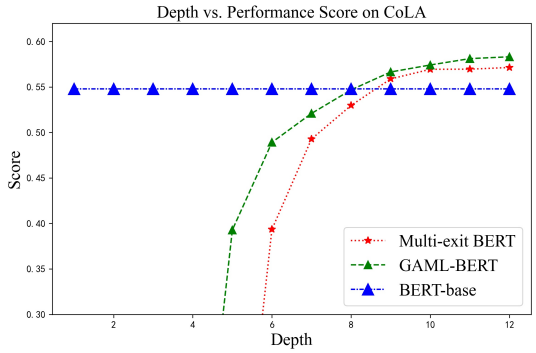
Although our work mainly works on NLP tasks, we also show that one can easily apply our GAML-BERT framework to image classification tasks in the Appendix.

6.6 Analysis

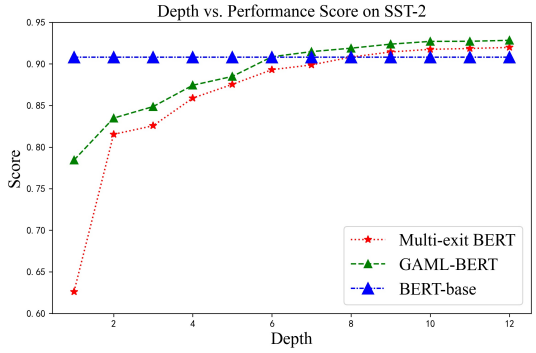
We now analyze more deeply the main takeaways from Table 2 and our experiments.

Our GAML-BERT method can improve the performances of early exiting, especially on shallow exits. To demonstrate our method's effectiveness and how it improves the shallow exits, we conduct budgeted exiting inference on each task and plot the relationship between the layer depth and the performance score in Figure 4. On CoLA, except that the first four exits have zero scores, all the exits of GAML-BERT outperform multi-exit BERT trained with Eq 1. Similar observations can be made on SST-2. Note that the performance margins on the shallow exits are more significant than those on the deep exits, showing that our model is effective in improving the shallow early exits' performances.

The ML strategies are beneficial. Table 2 reveals that the ML training objective provides the



(a) CoLA



(b) SST-2

Figure 4: The depth-score curves for different early exiting strategies. The x-axis is the depth of the exit (or the number of layers before entering this exit), the y-axis is the performance metrics following GLUE (Wang et al., 2018). We also add the performance of BERT-base for comparison.

best performances on GLUE in terms of knowledge distillation strategies. ML-BERT consistently outperforms FastBERT, and GAML-BERT outperforms all the baseline models, especially FastBERT trained with our gradient alignment method. These experimental results validate our findings in section 3 that mutual learning can help to improve the early exits to the greatest extend. The ML method imposes rich regularizations over all exits, thus improvig the performances of early exiting.

Our GA algorithm brings performance gains. From Table 2, we can see that our full model GAML-BERT, consistently outperforms the ML-BERT, sometimes with a large margin. In addition, we also combine FastBERT with our GA method, denoted by FastBERT-GA.[5] We can see

[5]Note that in FastBERT-GA, the KD loss terms are added to the CE loss terms, and the fine-tuning is done in a single stage, which is different from FastBERT's two-stage procedure.

α	ML-BERT	GAML-BERT
0.1	48.8	50.9
0.3	**49.2**	**51.1**
0.5	48.6	50.9
0.8	48.4	50.8

Table 3: The performances of the ML-BERT and GAML-BERT on CoLA with different values of α.

that FastBERT-GA also consistently outperforms FastBERT. These ablation results empirically prove that the hypothesis H1 is true and demonstrate the effectiveness of our GA method.

Our GA algorithm makes the model less sensitive to α. In this group of experiments, we show how changes in α (in Eq. 2) affect the performances of early exiting architectures, with or without our GA method. Table 3 reports the performance comparisons on the CoLA task. We can clearly see that GAML-BERT's performance changes are much more minor than those of ML-BERT, under different values of α. In addition, GAML-BERT outperforms ML-BERT under all values of α, showing that GA can effectively stabilize the training with KD and leads to better optimization.

How our GAML-BERT method affects the training time costs. Table 4 presents the training time costs for GAML-BERT compared with the original BERT and PABEE. Note that we adopt an early stopping mechanism for training. Thus the training time costs are measured by the average number of steps till early stopping. Firstly, although exits introduce extra parameters and extra time for training, early exiting architectures actually can reduce the training time. Intuitively, additional loss objectives can be regarded as additional parameter updating steps for lower layers, thus speeding up the model convergence. Secondly, ML-BERT converges earlier than PABEE, demonstrating the regularization functionality of KD. Third, GA further accelerates the convergences of ML-BERT. Intuitively, GA eliminates the conflicting factors of KD, and thus leading to faster convergences.

How our GAML-BERT method performs under different dynamic exiting strategies. Note that the main experimental results in Table 2 are obtained by adopting the PABEE's patience-based dynamic exiting strategies. However, our GAML-BERT model is off-the-shelf since it can be easily adapted to other exiting strategies. In Table 5, we

Method	#Param	Speed-up	MNLI	MRPC	SST-2
With DeeBERT's entropy-based exiting strategy					
DeeBERT	109M	1.86x	81.2	85.8	90.0
GAML-BERT (ours)	109M	1.93x	83.0	87.6	91.3
With Shallow-Deep's max-prob based exiting strategy					
Shallow-Deep	109M	1.91x	81.1	85.7	90.1
GAML-BERT (ours)	109M	1.94x	82.8	87.8	91.4

Table 5: Experimental results of GAML-BERT when using different early exiting strategies.

Method	Training time cost	
-	CoLA	SST-2
w/o early exiting	2300	4100
w PABEE	2100	3800
w FastBERT	2000	3300
w GAML-BERT	1800	2900

Table 4: Comparison of training time costs. The training time cost is measured by the number of steps till early stopping.

present the results of GAML-BERT under different dynamic exiting strategies. When inferencing with entropy-based strategy, GAML-BERT outperforms DeeBERT on all GLUE tasks. Moreover, similarly, GAML-BERT outperforms Shallow-Deep when it adopts the max probability strategy.

7 Conclusion

In this work, we propose GAML-BERT, a novel framework for improving PLMs' early exiting. Our contributions are three-fold. Firstly, we conduct a series of exploratory experiments, which shows that mutual knowledge distillation between a pair of exits can boost the performances of both. Following this observation, it is natural to apply mutual learning (ML), that is, asking all the exits to learn from each other, to enhance the performances of BERT early exiting. Second, we propose GA, a novel training method that aligns the gradients from knowledge distillation to cross-entropy losses. Experiments on the GLUE benchmark datasets show that our framework can improve PLMs' early exiting performances, especially under high latency requirements. In addition, our framework is off-the-shelf and can be adapted to various early exiting strategies.

References

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael R. Lyu, and Irwin King. 2020. Binarybert: Pushing the limit of BERT quantization. *CoRR*, abs/2012.15701.

Tolga Bolukbasi, J. Wang, O. Dekel, and Venkatesh Saligrama. 2017. Adaptive neural networks for efficient inference. In *ICML*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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.

Angela Fan, E. Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. *ArXiv*, abs/1909.11556.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Geoffrey E. Hinton, Oriol Vinyals, and J. Dean. 2015. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531.

Gao Huang, Danlu Chen, T. Li, Felix Wu, L. V. D. Maaten, and Kilian Q. Weinberger. 2017. Multi-scale dense convolutional networks for efficient prediction. *ArXiv*, abs/1703.09844.

Xiaoqi Jiao, Y. Yin, L. Shang, Xin Jiang, X. Chen, Linlin Li, F. Wang, and Qun Liu. 2020. Tinybert: Distilling bert for natural language understanding. *ArXiv*, abs/1909.10351.

Y. Kaya, Sanghyun Hong, and T. Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *ICML*.

Se-Hoon Kim, Amir Gholami, Zhewei Yao, M. W. Mahoney, and K. Keutzer. 2021. I-bert: Integer-only bert quantization. *ArXiv*, abs/2101.01321.

Alex. Krizhevsky. 2009. Learning multiple layers of features from tiny images.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. *ArXiv*, abs/1909.11942.

Weijie Liu, P. Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Q. Ju. 2020. Fastbert: a self-distilling bert with adaptive inference time. *ArXiv*, abs/2004.02178.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *NeurIPS*.

Mary Phuong and Christoph H. Lampert. 2019. Distillation-based training for multi-exit architectures. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1355–1364.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

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

Roy Schwartz, Gabi Stanovsky, Swabha Swayamdipta, Jesse Dodge, and N. A. Smith. 2020. The right tool for the job: Matching model and instance complexities. In *ACL*.

Dawei Sun, Anbang Yao, Aojun Zhou, and Hao Zhao. 2019a. Deeply-supervised knowledge synergy. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6990–6999.

S. Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019b. Patient knowledge distillation for bert model compression. In *EMNLP/IJCNLP*.

Christian Szegedy, V. Vanhoucke, S. Ioffe, Jonathon Shlens, and Z. Wojna. 2016. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. 2016. Branchynet: Fast inference via early exiting from deep neural networks. *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv: Computation and Language*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, L. Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *ArXiv*, abs/1706.03762.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *BlackboxNLP@EMNLP*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi 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 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 J. Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. In *ACL*.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and M. Zhou. 2020. Bert-of-theseus: Compressing bert by progressive module replacing. In *EMNLP*.

Z. Yang, Zihang Dai, Yiming Yang, J. Carbonell, R. Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*.

Ziqing Yang, Yiming Cui, ZhiPeng Chen, Wanxiang Che, T. Liu, S. Wang, and Guoping Hu. 2020. Textbrewer: An open-source knowledge distillation toolkit for natural language processing. In *ACL*.

Li Yuan, Francis E. H. Tay, Guilin Li, T. Wang, and Jiashi Feng. 2019. Revisit knowledge distillation: a teacher-free framework. *ArXiv*, abs/1909.11723.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. Ternarybert: Distillation-aware ultra-low bit bert. In *EMNLP*.

Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. 2020. Bert loses patience: Fast and robust inference with early exit. *ArXiv*, abs/2006.04152.

Michael Zhu and Suyog Gupta. 2018. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net.

Wei Zhu. 2021a. AutoRC: Improving BERT based relation classification models via architecture search. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and*

the 11th International Joint Conference on Natural Language Processing: Student Research Workshop, pages 33–43, Online. Association for Computational Linguistics.

Wei Zhu. 2021b. LeeBERT: Learned early exit for BERT with cross-level optimization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2968–2980, Online. Association for Computational Linguistics.

Wei Zhu. 2021c. MVP-BERT: Multi-vocab pre-training for Chinese BERT. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 260–269, Online. Association for Computational Linguistics.

Wei Zhu, Yuan Ni, Xiaoling Wang, and Guotong Xie. 2021. Discovering better model architectures for medical query understanding. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pages 230–237, Online. Association for Computational Linguistics.

A The over-thinking problem for BERT

In Figure 5, we demonstrate the over-thinking problem of BERT on SST-2 and CoLA. To obtain the performance of BERT-base's layer i, we insert a classifier at layer i and fine-tune BERT-base on the train set. We can see that the last layer does not obtain the best performances.

B A review of early exiting strategies

There are mainly three dynamic exiting strategies for BERT dynamic exiting. BranchyNet (Teerapittayanon et al., 2016), FastBERT (Liu et al., 2020), and DeeBERT (Xin et al., 2020) calculated the entropy of the prediction probability distribution as a proxy for the confidence of exiting classifiers to enable dynamic early exiting. Shallow-Deep Nets (Kaya et al., 2019) and RightTool (Schwartz et al., 2020) leveraged the softmax scores of predictions of exiting classifiers. If the score of a particular class is dominant and large enough, the model will exit. Recently, PABEE (Zhou et al., 2020) propose a patience-based dynamic exiting strategy analogous to early stopping model training. That is, if the exits' predictions remain unchanged for a predefined number of times (patience), the model will stop inference and exit. PABEE achieves SOTAs results for BERT early exiting.

In this work, we mainly adopt the PABEE's patience-based early exiting strategy. However, in ablation studies, we will show that our GAML-BERT framework can improve the inference performance of other exiting strategies.

C Hyperparameter settings

C.1 Hyperparameters for each task in the pilot experiments

Table 6 reports the important hyper-parameters of BERT for each task in the pilot experiments.

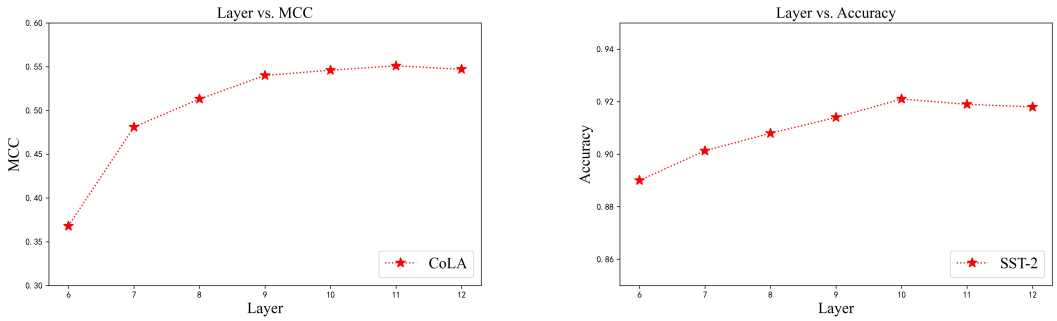
C.2 Hyper-parameters for each task in the main experiments

Table 7 reports the important hyper-parameters of GAML-BERT for each task. Note that our hyper-parameter search was done on the training set with cross-validation so that the GLUE benchmarks' dev set information was not revealed during training.

D GAML-BERT are effective for image classification

To demonstrate the effectiveness of GAML-BERT on the image classification task, we follow the experimental settings in Shallow-Deep (Kaya et al., 2019). We conduct experiments on two image classification datasets, CIFAR-10 and CIFAR-100 (Krizhevsky, 2009). The ResNet-56 model (He et al., 2016) serves as the backbone, and we compare GAML-BERT with PABEE, DBT from Phuong and Lampert (2019). We place an exiting classifier at every two convolutional layers. We set the batch size to 128 and use an SGD optimizer with a learning rate of 0.1.

Table 8 reports the results. GAML-BERT outperforms the original ResNet-56 on both tasks even when it provides 1.3x speed-up. Besides, it outperforms PABEE and DBT.



(a) CoLA (b) SST-2

Figure 5: The over-thinking phenomenon of BERT-base on the CoLA and SST-2 task. To obtain the performance of BERT-base's layer i, we insert a classifier at layer i and fine-tune BERT-base on the train set. The metric is MCC for CoLA, and ACC for SST-2. For CoLA, the highest score is obtained by layer 11. For SST-2, the highest score is obtained by layer 9, and deeper layers have lower performance scores.

Task	learning rate	warm-up steps (/epoch)	batch size	α
CoLA	1e-5	1.0	32	0.2
SST-2	2e-5	0.8	128	0.1

Table 6: Hyperparameter settings for the pilot experiments. We mainly tune learning rate, batch size, warm-up steps, and weight for the knowledge distillation loss term. Note that our hyper-parameter search was done on the training set with cross-validation so that the GLUE benchmarks' dev set information was not revealed during training.

Task	learning rate	warm-up steps (/epoch)	batch size	α
CoLA	5e-5	1.2	32	0.3
MNLI	2e-5	0.8	256	0.2
MRPC	2e-5	1.0	16	0.2
QNLI	1e-5	1.0	128	0.5
QQP	1e-5	0.8	256	0.3
RTE	5e-5	1.2	16	0.2
SST-2	5e-5	0.8	128	0.3

Table 7: Hyper-parameter settings for each task in the main experiments.

Method	CIFAR-10		CIFAR-100	
-	speed-up	Acc.	speed-up	Acc.
ResNet-56	1.00x	91.8	1.00x	68.6
PABEE	1.26x	91.8	1.22x	69.1
DBT	1.28	92.1	1.25x	69.3
GAML-BERT	1.30x	92.6	1.27x	69.7

Table 8: Experimental results of GAML-BERT when applied in the image classification tasks.