BinaryBERT: Pushing the Limit of BERT Quantization

Haoli Bai[1], Wei Zhang[2], Lu Hou[2], Lifeng Shang[2],
Jing Jin[3], Xin Jiang[2], Qun Liu[2], Michael Lyu[1], Irwin King[1]
[1] The Chinese University of Hong Kong
[2]Huawei Noah's Ark Lab, [3]Huawei Technologies Co., Ltd.
{hlbai, lyu, king}@cse.cuhk.edu.hk
{zhangwei379, houlu3, shang.lifeng, jinjing12, jiang.xin, qun.liu}@huawei.com

Abstract

The rapid development of large pre-trained language models has greatly increased the demand for model compression techniques, among which quantization is a popular solution. In this paper, we propose BinaryBERT, which pushes BERT quantization to the limit with weight binarization. We find that a binary BERT is hard to be trained directly than a ternary counterpart due to its complex and irregular loss landscapes. Therefore, we propose ternary weight splitting, which initializes the binary model by equivalent splitting from a half-sized ternary network. The binary model thus inherits the good performance of the ternary model, and can be further enhanced by fine-tuning the new architecture after splitting. Empirical results show that BinaryBERT has negligible performance drop compared to the full-precision BERT-base while being $24\times$ smaller, achieving the state-of-the-art results on GLUE and SQuAD benchmarks.

1 Introduction

Recent pre-trained language models have achieved remarkable performance improvement in various natural language tasks (Vaswani et al., 2017; Devlin et al., 2019). However, the improvement generally comes at the cost of increasing model size and computation, which limits the deployment of these huge pre-trained language models to edge devices. Various methods have been recently proposed to compress these models, such as knowledge distillation (Sanh et al., 2019; Sun et al., 2019; Jiao et al., 2020), pruning (Michel et al., 2019; Fan et al., 2019), low-rank approximation (Ma et al., 2019; Lan et al., 2020), weight-sharing (Dehghani et al., 2019; Lan et al., 2020), dynamic networks with adaptive depth and/or width (Liu et al., 2020; Hou et al., 2020; Xin et al., 2020; Zhou et al., 2020), and quantization (Zafrir et al., 2019; Shen et al., 2020; Fan et al., 2020; Zhang et al., 2020).
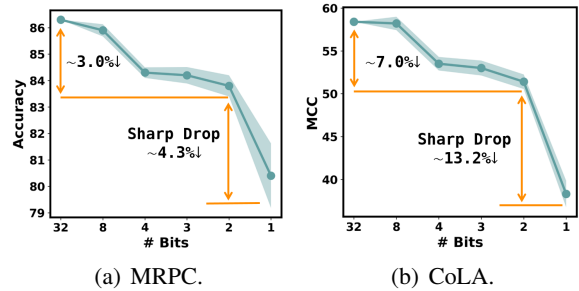

Figure 1: Performance of weight quantized BERT-base models under various bit-widths. The solid green line and shaded areas denote the mean and standard deviations over 10 random runs.

Among all these model compression approaches, quantization is a popular solution as it does not require designing a smaller model architecture. Instead, it compresses the model by replacing each 32-bit floating-point parameter with a low-bit fixed-point representation. Existing attempts tried to quantize pre-trained models (Zafrir et al., 2019; Shen et al., 2020; Fan et al., 2020) to even as low as ternary values (2-bit) with minor performance drop (Zhang et al., 2020). However, none of them achieve the binarization (1-bit). As the limit of quantization, weight binarization could bring up to $32\times$ reduction in model size. and replaces most floating-point multiplications into additions. In addition, quantizing the activations to 8-bit or 4-bit can further replace the floating-point addition into int8 and int4 addition, and decreases the energy burden and the area usage of the chips.

In this paper, we explore to binarize BERT parameters, pushing quantization to the limit. We find that directly training a binary network is rather challenging. According to Figure 1, there is a sharp performance drop when reducing weight bit-width from 2-bit to 1-bit, compared to other bit configurations. To explore the challenges of binarization, we analyze the loss landscapes of models under different precisions both qualitatively and quanti-

tatively. It is found that while the full-precision and ternary (2-bit) models enjoy relatively flat and smooth loss surfaces, the binary model suffers from a rather steep and complex landscape, which poses great challenges to the optimization.

Motivated by the above empirical observations, we propose *ternary weight splitting*, which takes the ternary model as a proxy to bridge the gap between the binary model and the full precision counterpart. Specifically, ternary weight splitting equivalently converts both the quantized and latent full-precision weights in a well-trained ternary model to initialize BinaryBERT. Therefore, Binary-BERT retains the good performance of the ternary model, and can be further refined on the new architecture. Neuron splitting has been previously used in computer vision (Chen et al., 2016; Wu et al., 2019). However, these methods are only designed for full-precision models. Recently Kim et al. (2019) propose to couple two binary activations into a ternary activation by shifting the bias of Batch Normalization layer. However, this method cannot be applied here because there is no batch normalization layer, and weight splitting is much more complex due to the equivalence constraint for both the quantized and latent full-precision weight.

Empirical results show that BinaryBERT split from a half-width ternary network is far better than a directly-trained a binary model on the original width. On GLUE benchmarks and SQuAD, our BinaryBERT has a negligible performance drop compared to the full-precision BERT-base model, while being $24\times$ smaller. Furthermore, our proposed approach also supports *adaptive splitting*. It can adaptively perform splitting on the most important ternary modules while leaving the rest as binary, based on efficiency constraints such as model size or floating-point operations (FLOPs). Therefore, our approach allows flexible sizes of binary models for various edge devices' demand.

2 Difficulty in Training Binary BERT

In this section, we show that it is challenging to train a binary BERT with conventional binarization approaches directly. Before diving into details, we first review the necessary backgrounds.

We follow the standard quantization-aware training procedure (Zhou et al., 2016). Specifically, given weight $\mathbf{w} \in \mathbb{R}^n$ (a.k.a latent full precision weights), each forward propagation quantizes it to $\hat{\mathbf{w}} = \mathcal{Q}(\mathbf{w})$ by some quantization function $\mathcal{Q}(\cdot)$,

and then compute the loss $\ell(\hat{\mathbf{w}})$ at $\hat{\mathbf{w}}$. During back propagation, we use $\nabla\ell(\hat{\mathbf{w}})$ to update latent full-precision weights $\mathbf{w}$ due to the non-differentiability of $\mathcal{Q}(\cdot)$, which is known as the straight-through estimator (Courbariaux et al., 2015).

Recent TernaryBERT (Zhang et al., 2020) adopts the TWN method (Li et al., 2016) to quantize the elements in $\mathbf{w}$ to three values $\{\pm\alpha, 0\}$. To avoid confusion, we use superscript t and b for the latent full-precision weight in ternary and binary models, respectively. Specifically, TWN ternarizes each element w_i^t in the ternary weight $\mathbf{w}^t$ as

$$\hat{w}_i^t = \mathcal{Q}(w_i^t) = \begin{cases} \alpha \cdot \text{sign}(w_i^t) & |w_i^t| \geq \Delta \\ 0 & |w_i^t| < \Delta \end{cases}, \quad (1)$$

where $\text{sign}(\cdot)$ is the sign function, $\Delta = \frac{0.7}{n}\|\mathbf{w}^t\|_1$ and $\alpha = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}|w_i^t|$ with $\mathcal{I} = \{i \mid |w_i^t| \geq 0\}$.

Binarization is first proposed in (Courbariaux et al., 2015) and has been extensively studied in the academia (Rastegari et al., 2016; Hubara et al., 2016; Liu et al., 2018). As a representative work, Binary-Weight-Network (BWN) (Hubara et al., 2016) has demonstrated superior advantages with a scaling parameter α. Specifically, BWN binarizes $\mathbf{w}^b$ element-wisely in the following way:

$$\hat{w}_i^b = \mathcal{Q}(w_i^b) = \alpha \cdot \text{sign}(w_i^b), \quad \alpha = \frac{1}{n}\|\mathbf{w}^b\|_1. \quad (2)$$

Despite the appealing properties of network binarization, we show that it is non-trivial to obtain a binary BERT with such approach.

### 2.1 Sharp Performance Drop with Weight Binarization

To study the performance drop caused by quantization, we train the BERT-base model with full-precision, 8-bit, 4-bit, 3-bit, 2-bit, and 1-bit weights on MRPC and CoLA from the GLUE benchmark (Wang et al., 2018). We use loss-aware weight quantization (LAQ) (Hou and Kwok, 2018) for 8/4-bit quantization, TWN (Li et al., 2016) for 2-bit ternarization and BWN (Hubara et al., 2016) and 1-bit binarization. For now we keep all activations in full-precision.

From Figure 1, the performance scores drop mildly from 32-bit to as low as 2-bit ($\sim 3.0\%$ ↓ on MRPC and $\sim 7.0\%$ ↓ CoLA). However, when the bit-width reduces from 2 to 1, there is a sharp drop of 4.3% and 13.2% on the two tasks, respectively. Similar patterns are also observed on other GLUE
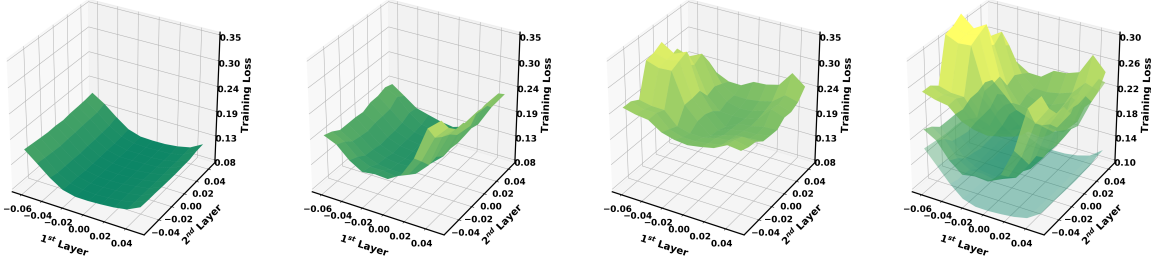
| (a) Full-precision Model. | (b) Ternary Model. | (c) Binary Model. | (d) All Together. |

Figure 2: Visualization of loss landscapes of the full-precision, ternary and binary models on MRPC. For (a), (b) and (c), we perturb the (latent) full-precision weights of the value layer in the 1st and 2nd Transformer layers, and compute their corresponding training loss. (d) shows the gap among the three surfaces by stacking them together.
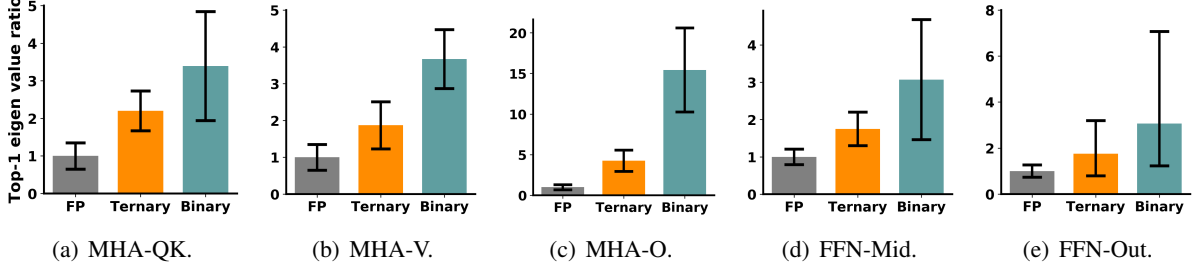
| (a) MHA-QK. | (b) MHA-V. | (c) MHA-O. | (d) FFN-Mid. | (e) FFN-Out. |

Figure 3: The top-1 eigenvalues of parameters at different Transformer parts of the full-precision (FP), ternary and binary BERT. For normalization, we report the ratio of eigenvalue between the ternary/binary models and the full-precision model. The error bar is estimated of all Transformer layers over different data mini-batches.

datasets. From the empirical results, weight binarization severely reduces the model capacity and harms the performance. Most current approaches stopped at the 2-bit quantization (Shen et al., 2020; Zadeh and Moshovos, 2020; Zhang et al., 2020). To further push weight quantization to the limit, a first step is to study the potential reasons behind the sharp drop from ternarization to binarization.

2.2 Exploring the Quantized Loss Landscape

Visualization To learn about the challenges behind the binarization, we visually compare the loss landscapes of the full-precision, ternary, and binary BERT models. Following (Nahshan et al., 2019), we extract parameters $\mathbf{w}_x, \mathbf{w}_y$ in the first two Transformer layers[1], and assign perturbations weighted by $(x, y) \in [-1, 1]^2$ as follows:

$$\tilde{\mathbf{w}}_x = \mathbf{w}_x + x \cdot \boldsymbol{\eta}_x, \quad \tilde{\mathbf{w}}_y = \mathbf{w}_y + y \cdot \boldsymbol{\eta}_y,$$

where $\boldsymbol{\eta}_x, \boldsymbol{\eta}_x$ are perturbation vectors whose elements equal the absolute mean values of $\mathbf{w}_x$ and $\mathbf{w}_y$, respectively. For each pair of $(x\boldsymbol{\eta}_x, y\boldsymbol{\eta}_y)$. We evaluate the corresponding training loss and plot the surface in Figure 2.

As can be seen, the full-precision model (Figure 2(a)) has the lowest overall training loss, and its

[1]For simplicity, here we use the parameters in the value layer in the multi-head attention.

loss landscape is flat and robust to the perturbation. For the ternary model (Figure 2(b)), despite the surface tilts up with larger perturbations, it is locally convex and is thus easy to optimize. This may also explain why the BERT model can be ternarized without severe accuracy drop (Zhang et al., 2020). However, the loss landscape of the binary model (Figure 2(c)) turns out to be more complex and the curvature is much steeper. A steeper curvature also reflects a higher sensitivity of the loss to binarization and can attribute to the difficulty in training. Meanwhile, the loss values of the binary BERT are also generally higher than the other two. By stacking the three landscapes together (Figure 2(d)), the loss surface of the binary BERT stands on the top with a clear margin with the other two.

Steepness Measurement To quantitatively measure the steepness of loss landscape, we start from a local minima $\mathbf{w}$ and apply the second order approximation to the curvature. According to the Taylor's expansion, the loss increase induced by quantizing $\mathbf{w}$ can be approximated by

$$\ell(\hat{\mathbf{w}}) - \ell(\mathbf{w}) \approx \boldsymbol{\epsilon}^\top \mathbf{H} \boldsymbol{\epsilon} \leq \lambda_{\max} \|\boldsymbol{\epsilon}\|^2,$$

where $\boldsymbol{\epsilon} = \mathbf{w} - \hat{\mathbf{w}}$ is the quantization noise, and $\lambda_{\max}$ is the largest eigenvalue of the Hessian $\mathbf{H}$ at $\mathbf{w}$. Note that the first-order term is skipped due to $\nabla \ell(\mathbf{w}) = 0$. Therefore, we take $\lambda_{\max}$ as a
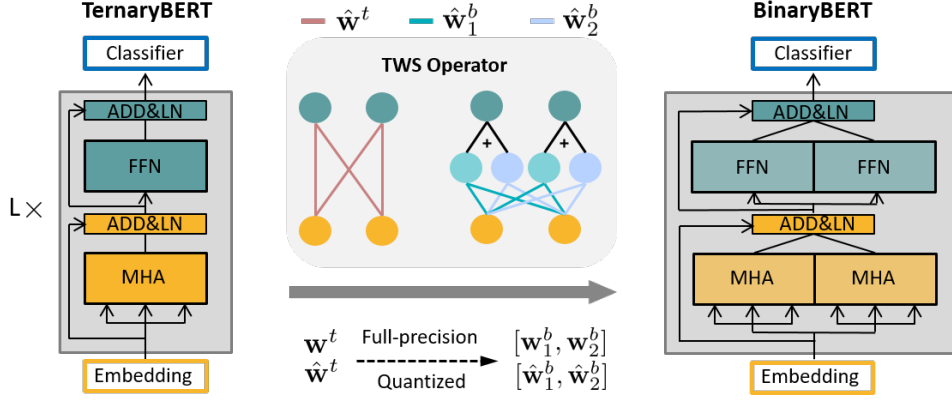
Figure 4: The overall workflow of training BinaryBERT. We first train a ternary BERT model, and then apply ternary weight splitting operator (Equation 4 and 5) to both the latent full-precision weights and quantized weights to initialize BinaryBERT, which is then fine-tuned for further refinement.

quantitative measurement for the steepness of the loss surface. Following (Shen et al., 2020) we adopt the power method to compute $\lambda_{\max}$. As it is computationally expensive to estimate $\mathbf{H}$ for all $\mathbf{w}$ in the network, we consider it separately as follows: the weight matrices in (1) the query/key layers (MHA-QK), (2) the value layer (MHA-V), (3) the output projection layer (MHA-O) in the multi-head attention, (4) the intermediate layer (FFN-Mid), and (5) the output layer (FFN-Out) in the feed-forward network. Note that we group the weights in the key and query layers as they are used together to calculate the attention scores.

From Figure 3, the largest eigenvalue of the binary model is increasingly larger with higher deviations compared to the full-precision baseline and the ternary model. For instance, the eigenvalues of MHA-O in the binary model is $\sim 15\times$ larger than the full-precision counterpart. Thus binarization leads to a large increase in the loss, making its landscape highly complex and irregular, which may pose more challenges to the optimization.

3 Proposed Method

3.1 Ternary Weight Splitting

Given the challenging loss landscape of binary BERT, we propose *ternary weight splitting* (TWS) that seeks to transfer the knowledge embedded in a trained ternary BERT to a binary BERT. As is shown in Figure 4, we first train a ternary BERT to convergence, and then split both the latent full-precision weight $\mathbf{w}^t$ and quantized $\hat{\mathbf{w}}^t$ to their binary counterparts $\mathbf{w}_1^b, \mathbf{w}_2^b$ and $\hat{\mathbf{w}}_1^b, \hat{\mathbf{w}}_2^b$ via the TWS operator, respectively. The weights after splitting are then used to initialize the binary model for further fine-tuning. To ensure the equivalency (i.e. the same output given the same input), we constrain

the latent full-precision and the quantized weights before and after splitting as:

$$\mathbf{w}^t = \mathbf{w}_1^b + \mathbf{w}_2^b, \quad \hat{\mathbf{w}}^t = \hat{\mathbf{w}}_1^b + \hat{\mathbf{w}}_2^b , \quad (3)$$

The solution of equation (3) is not unique. To solve equation (3), we first constrain the format of the latent full-precision weights after splitting $\mathbf{w}_1^b, \mathbf{w}_2^b$ to satisfy $\mathbf{w}^t = \mathbf{w}_1^b + \mathbf{w}_2^b$ as

$$w_{1,i}^b = \begin{cases} a \cdot w_i^t & \text{if } \hat{w}_i^t \neq 0 \\ b + w_i^t & \text{if } \hat{w}_i^t = 0, w_i^t > 0 \\ b & \text{otherwise} \end{cases} , \quad (4)$$

$$w_{2,i}^b = \begin{cases} (1-a)w_i^t & \text{if } \hat{w}_i^t \neq 0 \\ -b & \text{if } \hat{w}_i^t = 0, w_i^t > 0 \\ -b + w_i^t & \text{otherwise} \end{cases} , \quad (5)$$

where $a > 0$ and $b > 0$ are the variables to solve. Then by inserting equation (4) and (5) into $\hat{\mathbf{w}}^t = \hat{\mathbf{w}}_1^b + \hat{\mathbf{w}}_2^b$ in (3), we get

$$
\begin{aligned}
a &= \frac{\sum_{i \in \mathcal{I}} |w_i^t| + \sum_{j \in \mathcal{J}} |w_j^t| - \sum_{k \in \mathcal{K}} |w_k^t|}{2 \sum_{i \in \mathcal{I}} |w_i^t|}, \\
b &= \frac{\frac{n}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} |w_i^t| - \sum_{i=1}^n |w_i^t|}{2(|\mathcal{J}| + |\mathcal{K}|)},
\end{aligned} \quad (6)
$$

where we denote $\mathcal{I} = \{i \mid \hat{w}_i^t \neq 0\}$, $\mathcal{J} = \{j \mid \hat{w}_j^t = 0 \text{ and } w_j^t > 0\}$ and $\mathcal{K} = \{k \mid \hat{w}_k^t = 0 \text{ and } w_k^t < 0\}$. $|\cdot|$ denotes the cardinality of the set. Detailed derivation of Equation (6) is in Appendix A.

After splitting from the ternary model, the binary model inherits its performance on a new architecture with doubled width. Since the original ternary minimum may not hold in this new landscape, we perform further fine-tuning to look for a better solution. We dub the resulting model as BinaryBERT.

Quantization Details Following (Zhang et al., 2020), for each weight matrix in the Transformer layers, we use layer-wise ternarization (i.e. one scaling parameter for all elements in the weight matrix). After splitting, each of the two split matrices has its own scaling factor. For word embedding, we use row-wise ternarization (i.e. one scaling parameter for each row in the embedding). Similarly, each row has two scaling parameters after splitting.

We binarize the model parameters and simultaneously quantize all activations before the matrix multiplication, which could accelerate inference on specialized hardwares (Shen et al., 2020; Zafrir et al., 2019; Zhang et al., 2020). We use uniform quantization for the activations. Following (Zafrir et al., 2019; Zhang et al., 2020), we skip the quantization for all layer-normalization (LN) layers, skip connections, and bias as their calculation is negligible compared to dense matrix multiplication. We also skip the quantization for the last classification layer to avoid a large drop in accuracy.

Training with Knowledge Distillation Following (Jiao et al., 2020; Zhang et al., 2020), we use knowledge distillation to train both the ternary network and fine-tune the split binary network. The full-precision model acts as the teacher to supervise the quantized student model. For the ternary model, we use the two-stage distillation as (Jiao et al., 2020): the *intermediate-layer distillation* on the representations, and *prediction-layer distillation* on the logits. After splitting, we fine-tune the binary model with prediction-layer distillation.

3.2 Adaptive Splitting

Our proposed approach also supports *adaptive splitting* that can flexibly output binary models with different sizes. In practice, it may be inflexible to split all weight matrices in the ternary model on various resource-limited devices. To allow more efficient use of model capacity under various constraints $\mathcal{C}$ (e.g, model size and computational FLOPs), we can adaptively train a mixed-precision model (with a mixture of both binary and ternary weights) and then split the ternary weights into binary ones. Adaptive splitting thus enjoys the advantage of consistent arithmetic precision for all weight matrices, which is usually easier to deploy than the mixed-precision counterpart.

Intuitively, we assign ternary values to weight matrices that are more sensitive to quantization. The quantization sensitivity of the weight matrix is

empirically determined by the performance gain by not quantizing it compared to the fully-quantized counterpart (Details are in Appendix B.2.). We denote $\mathbf{u} \in \mathbb{R}_+^L$ as the sensitivity vector, where L is the total number of splittable weight matrices in all Transformer layers, the word embedding layer and the pooler layer. The cost vector $\mathbf{c} \in \mathbb{R}_+^L$ stores the additional increase of parameter size or FLOPs of each ternary weight matrix against a binary counterpart.

The assignment of splitting is determined by $\mathbf{s} \in \{0, 1\}^L$, where $s_i = 1$ means to ternarize the i-th weight matrix, and vice versa. The optimal assignment $\mathbf{s}^*$ (which maximizes the overall gain) can thus be obtained by solving the following combinatorial optimization problem:

$$\max_{\mathbf{s}} \quad \mathbf{u}^\top \mathbf{s} \qquad (7)$$
$$\text{s.t.} \quad \mathbf{c}^\top \mathbf{s} \leq \mathcal{C} - \mathcal{C}_0, \ \mathbf{s} \in \{0, 1\}^L,$$

where $\mathcal{C}_0$ is the baseline efficiency of the binary network. Dynamic programming can be used as the solver of Equation (7) to avoid the NP-hardness.

4 Experiments

In this section, we empirically evaluate the effectiveness of our proposed approach on the GLUE (Wang et al., 2018) and SQuAD (Rajpurkar et al., 2016, 2018) benchmark. We first introduce the experimental setup in Section 4.1, and then present the main experimental results on both benchmarks in Section 4.2. We compare to other state-of-the-arts in Section 4.3, and finally demonstrate the improvement by further fine-tuning the binary model after splitting in Section 4.4.

4.1 Experimental Setup

Dataset and Metrics The GLUE benchmark contains multiple natural language understanding tasks. We follow Devlin et al. (2019); Hou et al. (2020) to evaluate the performance on these tasks: Matthews correlation for CoLA, F1/accuracy for QQP, Spearman correlation for STS-B and accuracy for RTE, MRPC, SST-2, MNLI-m (matched) and MNLI-mm (mismatched). For machine reading comprehension on SQuAD, we report the EM (exact match) and F1 score.

Aside from the task performance, we also report the model size (MB) and computational FLOPs. For quantized operations, we follow (Zhou et al., 2016; Liu et al., 2018; Li et al., 2020) to count the bit-wise operations, i.e., the multiplication between

Table 1: Results on GLUE development set. "#Bits(W-E-A)" represents the number of bits for weights in the Transformer layers, word embedding, and activations. "DA" is short for data augmentation.

#	Quant	#Bits (W-E-A)	Size (MB)	FLOPs (G)	DA	MNLI -m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE
1	-	full-prec.	417.6	22.5	-	84.9/85.5	91.4	92.1	93.2	59.7	90.1	86.3	72.2
2	BWN	1-1-8	13.4	3.1	✗	84.2/84.0	91.1	90.7	92.3	46.7	86.8	82.6	68.6
3	TWS	1-1-8	16.5	3.1	✗	**84.2/84.7**	**91.2**	**91.5**	**92.6**	**53.4**	**88.6**	**85.5**	**72.2**
4	BWN	1-1-4	13.4	1.5	✗	83.5/83.4	90.9	90.7	**92.3**	34.8	84.9	79.9	**65.3**
5	TWS	1-1-4	16.5	1.5	✗	**83.9/84.2**	**91.2**	**90.9**	**92.3**	**44.4**	**87.2**	**83.3**	**65.3**
6	BWN	1-1-8	13.4	3.1	✓	84.2/84.0	91.1	91.2	92.7	54.2	88.2	**86.8**	70.0
7	TWS	1-1-8	16.5	3.1	✓	**84.2/84.7**	**91.2**	**91.6**	**93.2**	**55.5**	**89.2**	86.0	**74.0**
8	BWN	1-1-4	13.4	1.5	✓	83.5/83.4	90.9	91.2	92.5	51.9	87.7	85.5	70.4
9	TWS	1-1-4	16.5	1.5	✓	**83.9/84.2**	**91.2**	**91.4**	**93.7**	**53.3**	**88.6**	**86.0**	**71.5**

Table 2: Results on GLUE test set scored using the GLUE evaluation server.

#	Quant	#Bits (W-E-A)	Size (MB)	FLOPs (G)	DA	MNLI -m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Score
1	-	full-prec.	417.6	22.5	-	84.5/84.1	89.5	91.3	93.0	54.9	84.4	87.9	69.9	78.4
2	BWN	1-1-8	13.4	3.1	✗	83.3/83.4	88.9	**90.1**	92.3	38.1	81.2	**86.1**	63.1	74.7
3	TWS	1-1-8	16.5	3.1	✗	**84.1/83.6**	**89.0**	90.0	**93.1**	**50.5**	**83.4**	86.0	**65.8**	**76.8**
4	BWN	1-1-4	13.4	1.5	✗	83.5/82.5	**89.0**	**89.4**	92.3	26.7	78.9	84.2	59.9	72.4
5	TWS	1-1-4	16.5	1.5	✗	**83.6/82.9**	**89.0**	89.3	**93.1**	**37.4**	**82.5**	**85.9**	**62.7**	**74.7**
6	BWN	1-1-8	13.4	3.1	✓	83.3/83.4	88.9	**90.3**	91.3	48.4	**83.2**	**86.3**	66.1	76.4
7	TWS	1-1-8	16.5	3.1	✓	**84.1/83.5**	**89.0**	89.8	**91.9**	**51.6**	82.3	85.9	**67.3**	**76.8**
8	BWN	1-1-4	13.4	1.5	✓	83.5/82.5	**89.0**	**89.9**	92.0	45.0	81.9	85.2	64.1	75.5
9	TWS	1-1-4	16.5	1.5	✓	**83.6/82.9**	**89.0**	89.7	**93.1**	**47.9**	**82.9**	**86.6**	**65.8**	**76.4**

an m-bit number and an n-bit number approximately takes $mn/64$ FLOPs for a CPU with the instruction size of 64 bits.

Implementation We take DynaBERT (Hou et al., 2020) sub-networks as backbones as they offer both half-sized and full-sized models for easy comparison. We start from training a ternary model of width $0.5\times$ with the two-stage knowledge distillation introduced in Section 3.1. Then we split it into a binary model with width $1.0\times$, and perform further fine-tuning with prediction-layer distillation. Each training stage takes the same number of training epochs. In each stage, we train MNLI and QQP for 3 epochs and other smaller GLUE tasks for 6 epochs [2]. Following (Jiao et al., 2020; Hou et al., 2020; Zhang et al., 2020), we also try data augmentation on all GLUE tasks except for MNLI and QQP. More training details are in Appendix B.

We verify our ternary weight splitting (**TWS**) against vanilla binary training (**BWN**) with the same epochs for fair comparison.

Activation Quantization While BinaryBERT focuses on weight binarization, we also explore activation quantization in our implementation, which is beneficial for reducing the computation burden on specialized hardwares (Hubara et al., 2016;

Zhou et al., 2016; Zhang et al., 2020). Aside from the previously explored 8-bit quantization (Zhang et al., 2020; Shen et al., 2020), we further quantize activations to lower bits (e.g., 4-bit). Unlike (Zhang et al., 2020; Shen et al., 2020) which simply uses the minimum and maximum values to determine the quantization points, we use the Learned Step-size Quantization (LSQ) (Esser et al., 2019) to directly learn the quantized values. We find LSQ is better at dealing with activation outliers in BERT and achieves better empirical performance.
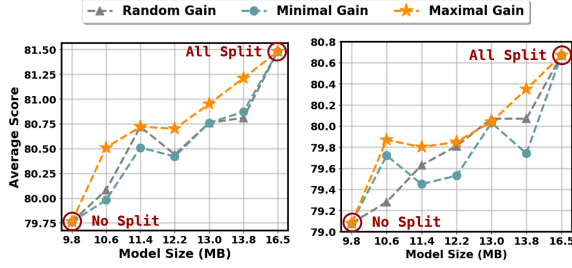
4.2 Experimental Results

4.2.1 Results on the GLUE Benchmark

The main results on the development set are shown in Table 1. For results without data augmentation (row #2-5), our ternary weight splitting method outperforms BWN with a clear margin [3]. For instance, on CoLA, ternary weight splitting achieves $6.7\% \uparrow$ and $9.6\% \uparrow$ with 8-bit and 4-bit activation quantization, respectively. While data augmentation (row 6-9) mostly improves each entry, our approach still overtakes BWN consistently. Furthermore, 4-bit activation quantization empirically benefits more from ternary weight splitting (row 4-5 and 8-9) compared to 8-bit activations (row

---

[2]Unlike results in Figure 1 that are trained with 3 epochs, here we prolong the training time for better performance.

[3]Note that DynaBERT only squeezes width in the Transformer layers but not the word embedding layer, thus the split binary model has a slightly larger size than BWN.

Figure 5: The adaptive average performance over six GLUE tasks of different splitting strategies.

2-3 and 6-7), demonstrating the potential of our approach in extremely low bit quantized models.

In Table 2, we also provide the results on the test set of GLUE benchmark. Similar to the observation in Table 1, our approach achieves consistent improvement on both 8-bit and 4-bit activation quantization compared to BWN.

Adaptive Splitting   The adaptive splitting in Section 3.2 supports the conversion of mixed ternary and binary models for more-fine-grained configurations. To verify its advantages, we compare it with two baseline strategies i) **Random Gain** that randomly selects weight matrices to split; and ii) **Minimal Gain** that splits the least important modules according to sensitivity. We report the average score over six tasks (QNLI, SST-2, CoLA, STS-B, MRPC and RTE) in Figure 5, where 9.8MB and 16.5MB correspond to split none and all modules in the model, respectively. As can be seen, our proposed adaptive splitting generally performs better than the other two baselines under each model size, indicating the effectiveness of maximizing the gain in adaptive splitting. In Appendix C, we provide detailed performance on the six tasks, together with the architecture visualization of adaptive splitting.

4.2.2   Results on SQuAD Benchmark

The results on the development set of SQuAD v1.1 and v2.0 are shown in Table 5. As can be seen, our proposed ternary weight splitting again outperforms BWN w.r.t both EM and F1 scores on both datasets. Similar to previous observations, 4-bit activation benefits more from the splitting approach with a larger gain in performance. For instance, our approach improves the EM score of 4-bit activation by 1.8% and 0.6% on SQuAD v1.1 and v2.0, respectively, both of which are higher than those of 8-bit activation.

Table 3: Development set results (EM/F1) on SQuAD.

Width	Quant	#Bits (W-E-A)	Size (MB)	FixOPs (G)	SQuAD v1.1	SQuAD v2.0
1.0×	-	*full-prec.*	417.6	22.5	82.6/89.7	75.1/77.5
1.0×	BWN	1-1-8	13.4	3.1	79.2/86.9	**73.6/76.6**
1.0×	TWS	1-1-8	16.5	3.1	**80.8/88.3**	73.6/76.5
1.0×	BWN	1-1-4	13.4	1.5	77.5/85.8	71.9/75.1
1.0×	TWS	1-1-4	16.5	1.5	**79.3/87.2**	**72.5/75.4**

4.3   Comparison with state-of-the-arts

Now we compare our proposed approach with a variety of state-of-the-art counterparts, including Q-BERT (Shen et al., 2020), GOBO (Zadeh and Moshovos, 2020), Quant-Noise (Fan et al., 2020) and TernaryBERT (Zhang et al., 2020). Aside from quantization, we also compare with other general compression approaches such as DistillBERT (Sanh et al., 2019), LayerDrop (Fan et al., 2019), TinyBERT (Jiao et al., 2020), ALBERT (Lan et al., 2020). The results are taken from the original papers, respectively. From Table 4, our proposed BinaryBERT has the smallest model size with the best performance among all quantization approaches. Compared to the full-precision model, our BinaryBERT retains competitive performance with a significant reduction of model size and computation. For example, we achieve more than $24\times$ compression ratio compared to BERT-base, with only $0.4\% \downarrow$ and $0.0\%/0.2\% \downarrow$ drop on MNLI-m on SQuAD v1.1, respectively.

Table 4: Comparison with other state-of-the-art methods on development set of MNLI-m and SQuAD v1.1.

Method	#Bits (W-E-A)	Size (MB)	Ratio ($\downarrow$)	MNLI -m	SQuAD v1.1
BERT-base	*full-prec.*	418	1.0	84.6	80.8/88.5
DistilBERT	*full-prec.*	250	1.7	81.6	79.1/86.9
LayerDrop-6L	*full-prec.*	328	1.3	82.9	-
LayerDrop-3L	*full-prec.*	224	1.9	78.6	-
TinyBERT-6L	*full-prec.*	55	7.6	82.8	79.7/87.5
ALBERT-E128	*full-prec.*	45	9.3	81.6	82.3/89.3
ALBERT-E768	*full-prec.*	120	3.5	82.0	81.5/88.6
Quant-Noise	PQ	11.0	38	83.6	-
Q-BERT	2/4-8-8	53	7.9	83.5	79.9/87.5
Q-BERT	2/3-8-8	46	9.1	81.8	79.3/87.0
Q-BERT	2-8-8	28	15.0	76.6	69.7/79.6
GOBO	3-4-32	43	9.7	83.7	-
GOBO	2-2-32	28	15.0	71.0	-
TernaryBERT	2-2-8	28	15.0	83.5	79.9/87.4
BinaryBERT	1-1-8	17	24.6	84.2	80.8/88.3
BinaryBERT	1-1-4	17	24.6	83.9	79.3/87.2

4.4   Further Improvement after Splitting

Finally, we demonstrate the performance gain by refining the binary model on the new architecture. We evaluate the performance gain after splitting from a half-width TWN model on the development set of SQuAD v1.1, v2.0, MNLI-m and MRPC. The results are shown in Table 5. As can be seen,

further fine-tuning brings consistent improvement on both 8-bit and 4-bit activation for all datasets, indicating that a better solution of the binary network can be found by the proposed ternary weight splitting method.

Table 5: The performance gain by fine-tuning the binary model after splitting.

Width	Quant	#Bits (W-E-A)	SQuAD v1.1	SQuAD v2.0	MNLI -m	MRPC
0.5×	TWN	2-2-8	80.3/87.9	72.9/76.0	84.1	83.6
1.0×	TWS	1-1-8	**80.8/88.3**	**73.6/76.5**	**84.2**	**85.5**
0.5×	TWN	1-1-4	78.0/86.4	72.1/75.1	83.7	83.1
1.0×	TWS	1-1-4	**79.3/87.2**	**72.5/75.4**	**83.9**	**83.3**

5   Conclusion

In this paper, we propose BinaryBERT that pushes the BERT quantization to the limit. As a result of the steep and complex loss landscape, we find directly training a BinaryBERT is hard with a large performance drop. We therefore propose a ternary weight splitting that split a trained ternary BERT to initialize the binary BERT. The binary model thus immediately inherits the good performance, based on which fine-tuning can bring further refinement. Empirical results show that our proposed method significantly outperforms vanilla binary training, and the resulting BinaryBERT achieves state-of-the-art performance on BERT compression.

References

T. Chen, I. Goodfellow, and J. Shlens. 2016. Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*.

M. Courbariaux, Y. Bengio, and J. David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131.

M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser. 2019. Universal transformers. In *International Conference on Learning Representations*.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, pages 4171–4186.

S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha. 2019. Learned step size quantization. In *International Conference on Learning Representations*.

A. Fan, E. Grave, and A. Joulin. 2019. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*.

A. Fan, P. Stock, B. Graham, E. Grave, R. Gribonval, H. Jegou, and A. Joulin. 2020. Training with quantization noise for extreme model compression. Preprint arXiv:2004.07320.

L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. In *Advances in Neural Information Processing Systems*, volume 33.

L. Hou and J. T. Kwok. 2018. Loss-aware weight quantization of deep networks. In *International Conference on Learning Representations*.

I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. 2016. Binarized neural networks. In *Advances in neural information processing systems*, volume 29, pages 4107–4115.

X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. 2020. Tinybert: Distilling bert for natural language understanding. In *Findings of Empirical Methods in Natural Language Processing*.

H. Kim, K. Kim, J. Kim, and J. Kim. 2019. Binaryduo: Reducing gradient mismatch in binary activation network by coupling binary activations. In *International Conference on Learning Representations*.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

F. Li, B. Zhang, and B. Liu. 2016. Ternary weight networks. Preprint arXiv:1605.04711.

Y. Li, X. Dong, and W. Wang. 2020. Additive powers-of-two quantization: A non-uniform discretization for neural networks. In *International Conference on Learning Representations*.

W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, and Q. Ju. 2020. Fastbert: a self-distilling bert with adaptive inference time. In *Annual Meeting of the Association for Computational Linguistics*.

Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K. Cheng. 2018. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *European Conference on Computer Vision*, pages 722–737.

X. Ma, P. Zhang, S. Zhang, N. Duan, Y. Hou, D. Song, and M. Zhou. 2019. A tensorized transformer for language modeling. In *Advances in Neural Information Processing Systems*.

P. Michel, O. Levy, and G. Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*, pages 14014–14024.

Y. Nahshan, B. Chmiel, C. Baskin, E. Zheltonozhskii, R. Banner, A. M. Bronstein, and A. Mendelson. 2019. Loss aware post-training quantization. Preprint arXiv:1911.07190.

P. Rajpurkar, R. Jia, and P. Liang. 2018. Know what you don't know: Unanswerable questions for squad. Preprint arXiv:1806.03822.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. Preprint arXiv:1606.05250.

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542.

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

S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *AAAI Conference on Artificial Intelligence*.

S. Sun, Y. Cheng, Z. Gan, and J. Liu. 2019. Patient knowledge distillation for bert model compression. In *Conference on Empirical Methods in Natural Language Processing*, pages 4314–4323.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. Preprint arXiv:1804.07461.

L. Wu, D. Wang, and Q. Liu. 2019. Splitting steepest descent for growing neural architectures. In *Advances in Neural Information Processing Systems*, volume 32, pages 10656–10666.

J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. In *Annual Meeting of the Association for Computational Linguistics*.

A. H. Zadeh and A. Moshovos. 2020. Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference. Preprint arXiv:2005.03842.

O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat. 2019. Q8bert: Quantized 8bit bert. Preprint arXiv:1910.06188.

W. Zhang, L. Hou, Y. Yin, L. Shang, X. Chen, X. Jiang, and Q. Liu. 2020. Ternarybert: Distillation-aware ultra-low bit bert. In *Empirical Methods in Natural Language Processing*.

S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. Preprint arXiv:1606.06160.

W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei. 2020. Bert loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*.

A Derivation of Equation (6)

In this section, we show the procedure to derive a and b according to $\hat{\mathbf{w}}^t = \hat{\mathbf{w}}_1^b + \hat{\mathbf{w}}_2^b$. According to the BWN quantizer introduced in Section 2, we have

$$\hat{w}_{1,i}^b = \alpha_1 \text{sign}(w_{1,i}^b),$$

where

$$\alpha_1 = \frac{1}{n}\Big[\sum_{i\in\mathcal{I}}|aw_i^t| + \sum_{i\in\mathcal{J}}|w_j^t + b| + \sum_{i\in\mathcal{K}}|b|\Big].$$

Similarly,

$$\hat{w}_{2,i}^b = \alpha_2 \text{sign}(w_{2,i}^b),$$

where

$$\alpha_2 = \frac{1}{n}\Big[\sum_{i\in\mathcal{I}}|(1-a)w_i^t| + \sum_{j\in\mathcal{J}}|-b| + \sum_{k\in\mathcal{K}}|w_k^t - b|\Big].$$

For those $\hat{w}_i^t = \hat{w}_{1,i}^b + \hat{w}_{2,i}^b = 0$, we have

$$\frac{1}{n}\Big[\sum_{i\in\mathcal{I}}|aw_i^t| + \sum_{j\in\mathcal{J}}|w_j^t + b| + \sum_{k\in\mathcal{K}}|b|\Big]$$
$$= \frac{1}{n}\Big[\sum_{i\in\mathcal{I}}|(1-a)w_i^t| + \sum_{j\in\mathcal{J}}|-b| + \sum_{k\in\mathcal{K}}|w_k^t - b|\Big].$$

This can be simplified to

$$a\sum_{i\in\mathcal{I}}|w_i^t| + \sum_{j\in\mathcal{J}}|w_j^t| = (1-a)\sum_{i\in\mathcal{I}}|w_i^t| + \sum_{k\in\mathcal{K}}|w_k^t|,$$

which gives the solution of a as

$$a = \frac{\sum_{i\in\mathcal{I}}|w_i^t| + \sum_{j\in\mathcal{J}}|w_j^t| - \sum_{k\in\mathcal{K}}|w_k^t|}{2\sum_{i\in\mathcal{I}}|w_i^t|}.$$

For $\hat{w}_i^t \neq 0$, from $\hat{w}_i^t = \hat{w}_{1,i}^b + \hat{w}_{2,i}^b$, we have

$$\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}|w_i^t| = \alpha_1 + \alpha_2$$
$$= \frac{1}{n}\Big[\sum_{i\in\mathcal{I}}|aw_i^t| + \sum_{j\in\mathcal{J}}|w_j^t + b| + \sum_{k\in\mathcal{K}}|b|\Big]$$
$$+ \frac{1}{n}\Big[\sum_{i\in\mathcal{I}}|(1-a)w_i^t| + \sum_{j\in\mathcal{J}}|-b| + \sum_{k\in\mathcal{K}}|w_k^t - b|\Big]$$
$$= \frac{1}{n}\Big[\sum_{i\in\mathcal{I}}|w_i^t| + \sum_{j\in\mathcal{J}}|w_j^t| + \sum_{k\in\mathcal{K}}|w_k^t|$$
$$+ 2\sum_{j\in\mathcal{J}}|b| + 2\sum_{k\in\mathcal{K}}|b|\Big]$$
$$= \frac{1}{n}\Big[\sum_{i=1}^{n}|w_i^t| + 2(|\mathcal{J}| + |\mathcal{K}|)\cdot b\Big].$$

Thus the solution for b is

$$b = \frac{\frac{n}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}|w_i^t| - \sum_{i=1}^{n}|w_i^t|}{2(|\mathcal{J}| + |\mathcal{K}|)}.$$

B Implementation Details

B.1 Hyper-parameter Settings

We first perform the two-stage knowledge distillation (i.e. intermediate-layer distillation and prediction-layer distillation) on the ternary model, and then perform ternary weight splitting followed by fine-tuning with only prediction-layer distillation after the splitting. The initial learning rate is set as 5×10^{-5} for the intermediate-layer distillation, and 2×10^{-5} for the prediction-layer distillation, both of which linearly decay to 0 at the end of training. We conduct experiments on GLUE tasks both without and with data augmentation (DA) except for MNLI and QQP due to their limited performance gain. The running epochs for MNLI and QQP are set to 3, and 6 for the rest tasks if without DA and 1 otherwise. For the rest hyper-parameters, we follow the default setting in (Devlin et al., 2019). The detailed hyper-parameters are summarized in Table 6.

Table 6: Hyper-parameters for training BinaryBERT on the GLUE benchmark. "Int. Distil." and "Pred. Distil." represents the intermediate-layer distillation and prediction-layer distillation, respectively.

	BinaryBERT		
	Int. Dstil. (Ternary)	Pred. Distil. (Ternary)	Split (Binary)
Batch Size	32	32	32
Sequence Length	128	128	128
Learning rate (LR)	5e-5	2e-5	2e-5
LR Decay	Linear	Linear	Linear
Warmup portion	0.1	0.1	0.1
Weight Decay	1e-2	1e-2	1e-2
Gradient Clipping	1	1	1
Dropout	0.1	0.1	0.1
Epochs w/o DA -other dataserts	6	6	6
Epochs w DA -other dataserts	1	1	1
Epochs w/o DA -MNLI, QQP	3	3	3

B.2 Detailed Procedure of Adaptive Splitting

As mentioned in Section 3.2, the adaptive splitting requires to first estimate the quantization sensitivity $\mathbf{u}$. We study the sensitivity in two aspects: the Transformer parts, and the Transformer layers. For Transformer parts, we still categorize the weights in Transformer layers into five parts as in Section 2.2. For each part, we quantize all other parts in all Transformer layers but leave the one under investigation un-quantized, and calculate the performance

(a) Transformer parts.
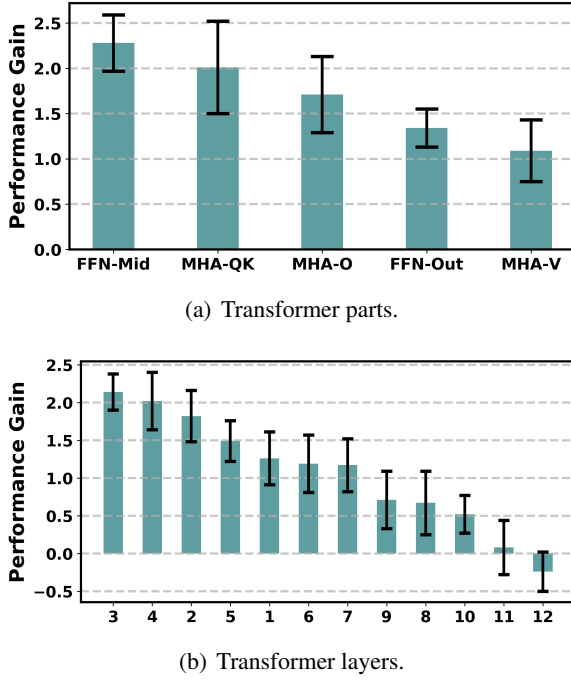

(b) Transformer layers.

Figure 6: The performance gain of different Transformer parts and layers in descending order. All numbers are averaged by 10 random runs with standard deviations reported.

gain compared to a fully-quantized model (Figure 6(a)). Similarly, for each Transformer layer, we quantize all layers but leave the one under investigation un-quantized, and calculate the performance gain compared to the fully-qauntized baseline (Figure 6(b)). As can be seen, for Transformer parts, the FFN-Mid and MHA-Q/K rank in the first and second place. In terms of Transformer layers, shallower layers are more sensitive to quantization than the deeper ones.

After that, we calculate the average gain by dividing the performance gain by the number of parameters in that part or layer. Therefore, we are able to measure the quantization sensitivity of the ith Transformer part in the jth Transformer layer by summing their average performance gain together. A larger value gain indicates a higher sensitivity score and thus a higher priority for splitting. Similarly, we also calculate the average performance gain in the word embedding and the pooler layer as their sensitivities, and use them as well as the Transformer weights' sensitivities above for the adaptive splitting in Section 3.2.

C More Empirical Results

C.1 Detailed Results of Adaptive Splitting

The detailed comparison of our adaptive splitting strategy against the random strategy (Rand.) and

minimal gain strategy (Min.) under different model size are shown in Table 7 and Table 8. It can be found that for both 8-bit and 4-bit activation quantization, our strategy that splits the most sensitive modules mostly performs the best on average under various model sizes.

C.2 Architecture Visualization

We further visualize the architectures after adaptive splitting on MRPC in Figure 7. For clear presentation, we merge all splittable parameters in each Transformer layer. As the baseline, 9.8MB refers to no splitting, while 16.5MB refers to splitting all splittable parameters in the model. According to Figure 7, with the increasing model size, shallower layers are more preferred for splitting than deeper layers, which is consistent to the findings in Figure 6.
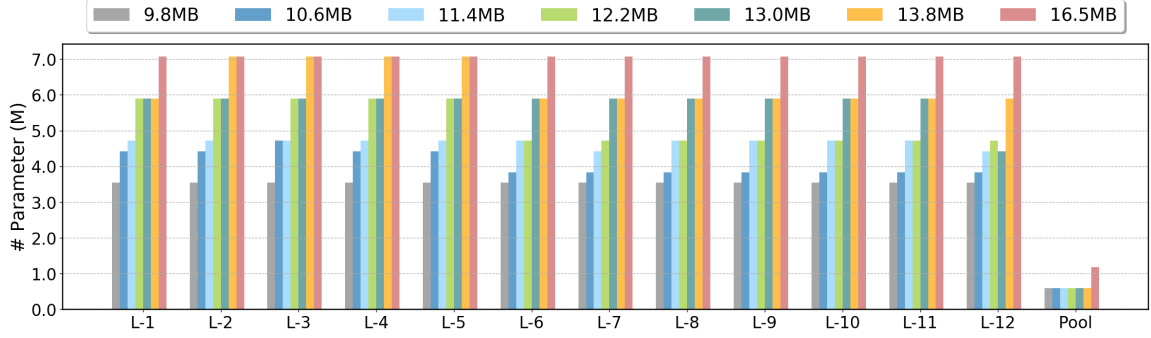
Figure 7: The architecture visualization for adaptive splitting on MRPC. The y-axis records the number of parameters split in each layer instead of the storage.

Table 7: Results on GLUE development set for adaptive splitting with 8-bit activation quantization.

Size (MB)	Strategy	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
	Max.	91.0	92.7	53.7	88.0	86.5	71.1	80.5
10.6	Rand.	90.8	92.7	53.3	88.2	85.5	70.0	80.1
	Min.	91.1	93.1	52.8	88.2	85.3	69.3	80.0
	Max.	91.0	93.0	54.6	88.4	86.3	71.1	80.7
11.4	Rand.	91.0	92.9	54.7	88.4	86.5	70.8	80.7
	Min.	91.0	93.0	53.8	88.3	85.5	71.5	80.5
	Max.	91.0	92.9	53.8	88.6	86.8	71.1	80.7
12.2	Rand.	91.1	92.9	54.1	88.5	86.0	71.8	80.4
	Min.	91.1	92.7	53.5	88.5	85.3	71.5	80.4
	Max.	91.1	93.1	56.1	88.6	86.1	70.8	81.0
13.0	Rand.	91.2	92.9	54.1	88.4	86.0	71.8	80.8
	Min.	91.2	92.8	54.8	88.5	85.1	72.2	80.8
	Max.	91.4	92.9	55.5	88.7	86.3	72.6	81.2
13.8	Rand.	91.5	92.9	54.7	88.5	85.0	72.2	80.8
	Min.	91.1	93.0	55.4	88.5	85.8	71.5	80.9

Table 8: Results on GLUE development set for adaptive splitting with 4-bit activation quantization.

Size (MB)	Strategy	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
	Max.	90.9	92.7	53.5	87.5	84.6	70.0	79.9
10.6	Rand.	91.1	92.7	51.3	87.6	84.8	68.2	79.3
	Min.	90.6	92.6	51.7	87.4	85.3	70.8	79.7
	Max.	91.1	92.6	52.1	87.7	85.3	70.0	79.8
11.4	Rand.	90.8	92.8	51.7	87.5	84.6	70.4	79.6
	Min.	90.9	92.8	50.9	87.6	85.3	69.4	79.5
	Max.	90.9	92.9	52.2	87.6	85.1	70.4	79.9
12.2	Rand.	91.2	93.0	52.0	87.6	85.1	70.0	79.8
	Min.	90.9	92.7	50.8	87.6	84.8	70.4	79.5
	Max.	91.3	92.9	53.4	87.8	85.3	69.7	80.1
13.0	Rand.	91.3	93.0	52.9	87.8	85.8	69.7	80.1
	Min.	91.1	92.8	52.6	87.7	86.3	69.7	80.0
	Max.	91.3	92.8	53.6	88.0	85.8	70.8	80.4
13.8	Rand.	91.3	92.9	52.3	87.7	85.1	71.1	80.1
	Min.	91.1	93.1	51.5	87.9	84.8	70.0	79.7