BinaryBERT: Pushing the Limit of BERT Quantization

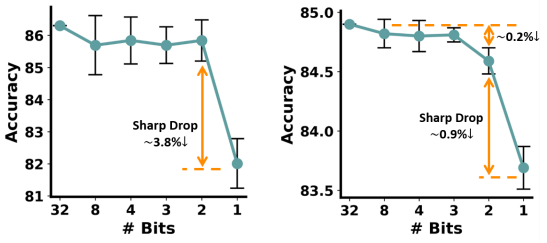
Haoli Bai[1], **Wei Zhang**[2], **Lu Hou**[2], **Lifeng Shang**[2],
Jing Jin[3], **Xin Jiang**[2], **Qun Liu**[2], **Michael Lyu**[1], **Irwin King**[1]
[1] The Chinese University of Hong Kong
[2]Huawei Noah's Ark Lab, [3]Huawei Technologies Co., Ltd.
{hlbai, lyu, king}@cse.cuhk.edu.hk
{zhangwei379, 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 by 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 landscape. Therefore, we propose ternary weight splitting, which initializes BinaryBERT by equivalently splitting from a half-sized ternary network. The binary model thus inherits the good performance of the ternary one, and can be further enhanced by fine-tuning the new architecture after splitting. Empirical results show that our BinaryBERT has only a slight performance drop compared with the full-precision model while being $24\times$ smaller, achieving the state-of-the-art compression results on the 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; Huang et al., 2021), dynamic networks with adaptive depth and/or width (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).



(a) MRPC. (b) MNLI-m.

Figure 1: Performance of quantized BERT with varying weight bit-widths and 8-bit activation. We report the mean results with standard deviations from 10 seeds on MRPC and 3 seeds on MNLI-m, respectively.

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 try 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 achieves the binarization (1-bit). As the limit of quantization, weight binarization could bring at most $32\times$ reduction in model size and replace most floating-point multiplications with additions. Moreover, quantizing activations to 8-bit or 4-bit further replaces the floating-point addition with int8 and int4 addition, decreasing the energy burden and the area usage on chips (Courbariaux et al., 2015).

In this paper, we explore to binarize BERT parameters with quantized activations, pushing BERT 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

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pages 4334–4348
August 1–6, 2021.

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 quantitatively. 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 and full-precision models. Specifically, ternary weight splitting equivalently converts both the quantized and latent full-precision weights in a well-trained ternary model to initialize BinaryBERT. Therefore, BinaryBERT retains the good performance of the ternary model, and can be further refined on the new architecture. While neuron splitting is previously studied (Chen et al., 2016; Wu et al., 2019) for full-precision network, our ternary weight splitting is much more complex due to the additional equivalence requirement of quantized weights. Furthermore, the proposed BinaryBERT 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' demands.

Empirical results show that BinaryBERT split from a half-width ternary network is much better than a directly-trained binary model with the original width. On the GLUE and SQuAD benchmarks, our BinaryBERT has only a slight performance drop compared to the full-precision BERT-base model, while being $\mathbf{24}\times$ smaller. Moreover, BinaryBERT with the proposed importance-based adaptive splitting also outperforms other splitting criteria across a variety of model sizes.

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 computes 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) follows Ternary-Weight-Network (TWN) (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 weights and quantized weights 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 $\mathcal{I} = \{i \mid \hat{w}_i^t \neq 0\}$.

Binarization. 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) binarizes $\mathbf{w}^b$ elementwisely with a scaling parameter α as follows:

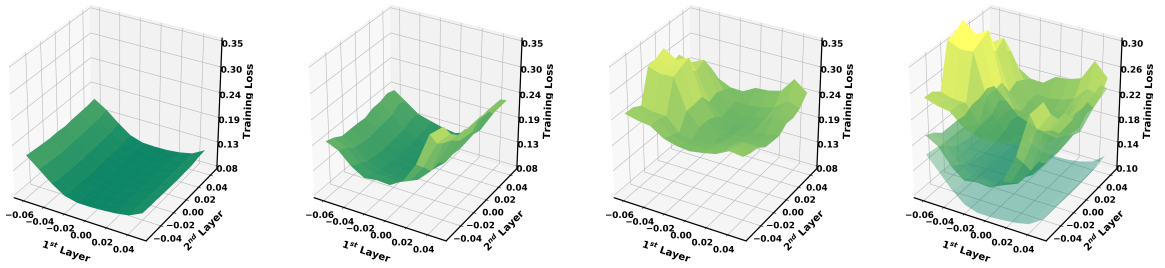
$$\hat{w}_i^b = \mathcal{Q}(w_i^b) = \alpha \cdot \text{sign}(w_i^b), \ \alpha = \frac{1}{n}\|\mathbf{w}^b\|_1. \quad (2)$$

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

2.1 Sharp Performance Drop with Weight Binarization

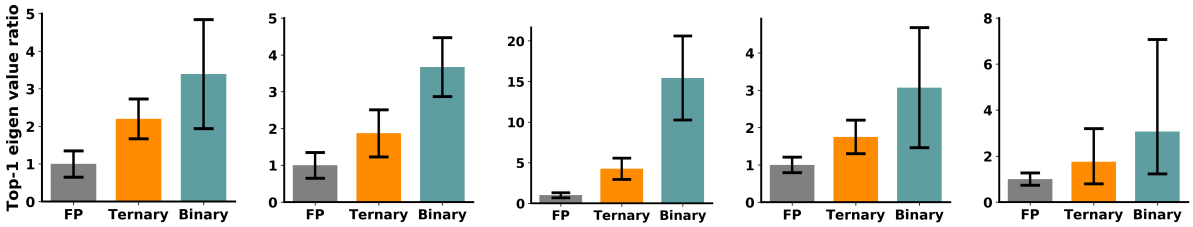
To study the performance drop of BERT quantization, we train the BERT model with full-precision, {8,4,3,2,1}-bit weight quantization and 8-bit activations on MRPC and MNLI-m from the GLUE benchmark (Wang et al., 2018) [1]. We use loss-aware weight quantization (LAQ) (Hou and Kwok, 2018) for 8/4/3-bit weight quantization, TWN (Li et al., 2016) for weight ternarization and BWN (Hubara et al., 2016) for weight binarization. Meanwhile, we adopt 8-bit uniform quantization for activations. We follow the default experimental settings detailed in Section 4.1 and Appendix C.1.

[1] We conduct more experiments on other GLUE datasets and with different settings in Appendix C.1, and find similar empirical results to MRPC and MNLI-m here.



(a) Full-precision Model. (b) Ternary Model. (c) Binary Model. (d) All Together.

Figure 2: Loss landscapes visualization 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 easy comparison, 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.

From Figure 1, the performance drops mildly from 32-bit to as low as 2-bit, i.e., around $0.6\% \downarrow$ on MRPC and $0.2\% \downarrow$ on MNLI-m. However, when reducing the bit-width to one, the performance drops sharply, i.e, $\sim 3.8\% \downarrow$ and $\sim 0.9\% \downarrow$ on the two tasks, respectively. Therefore, weight binarization may severely harm the performance, which may explain why most current approaches stop at 2-bit weight 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 first visually compare the loss landscapes of full-precision, ternary, and binary BERT models. Following (Nahshan et al., 2019), we extract parameters $\mathbf{w}_x, \mathbf{w}_y$ from the value layers[2] of multi-head attention in the first two Transformer layers, and assign the following perturbations on parameters:

$$\tilde{\mathbf{w}}_x = \mathbf{w}_x + x \cdot \mathbf{1}_x, \quad \tilde{\mathbf{w}}_y = \mathbf{w}_y + y \cdot \mathbf{1}_y, \quad (3)$$

where $x \in \{\pm 0.2\bar{w}_x, \pm 0.4\bar{w}_x, ..., \pm 1.0\bar{w}_x\}$ are perturbation magnitudes based the absolute mean value $\bar{w}_x$ of $\mathbf{w}_x$, and similar rules hold for y. $\mathbf{1}_x$ and $\mathbf{1}_y$ are vectors with all elements being 1. For each pair of (x, 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 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 looks 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 both higher and more complex. 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. The steep curvature of loss surface reflects a higher sensitivity to binarization, which attributes to the training difficulty.

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

[2]We also extract parameters from other parts of the Transformer in Appendix C.2, and the observations are similar.

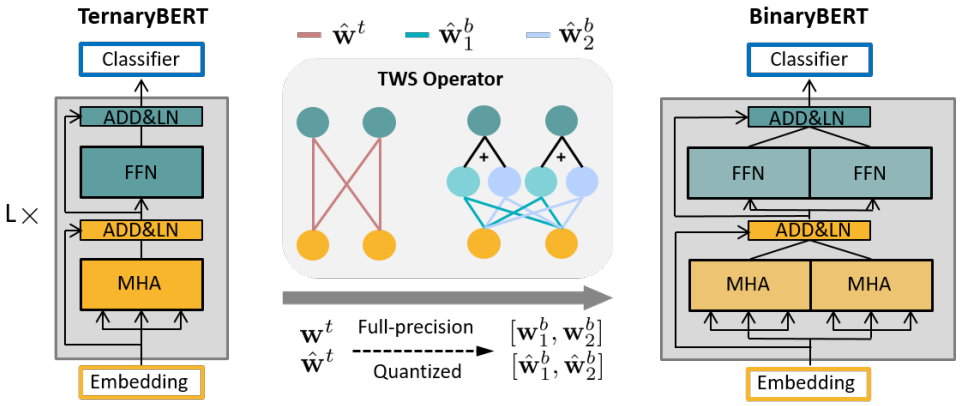


Figure 4: The overall workflow of training BinaryBERT. We first train a half-sized ternary BERT model, and then apply ternary weight splitting operator (Equations (6) and (7)) to obtain the latent full-precision and quantized weights as the initialization of the full-sized BinaryBERT. We then fine-tune BinaryBERT for further refinement.

$\mathbf{w}$ can be approximately upper bounded by

$$\ell(\hat{\mathbf{w}}) - \ell(\mathbf{w}) \approx \boldsymbol{\epsilon}^\top \mathbf{H} \boldsymbol{\epsilon} \leq \lambda_{\max} \|\boldsymbol{\epsilon}\|^2, \quad (4)$$

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 $\nabla \ell(\mathbf{w}) = 0$. Thus we take $\lambda_{\max}$ as a 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 them separately as follows: (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 key and query layers as they are used together to calculate the attention scores.

From Figure 3, the top-1 eigenvalues of the binary model are higher both on expectation and standard deviation compared to the full-precision baseline and the ternary model. For instance, the top-1 eigenvalues of MHA-O in the binary model are $\sim 15\times$ larger than the full-precision counterpart. Therefore, the quantization loss increases of full-precision and ternary model are tighter bounded than the binary model in Equation (4). The highly complex and irregular landscape by binarization thus poses 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 exploits the flatness of ternary loss landscape as the optimization proxy of the binary model. As is shown in Figure 4, we first train the half-sized 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*. To inherit the performance of the ternary model after splitting, the TWS operator requires the splitting equivalency (i.e., the same output given the same input):

$$\mathbf{w}^t = \mathbf{w}_1^b + \mathbf{w}_2^b, \quad \hat{\mathbf{w}}^t = \hat{\mathbf{w}}_1^b + \hat{\mathbf{w}}_2^b \ . \quad (5)$$

While solution to Equation (5) is not unique, we constrain 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

$$w_{1,i}^b = \begin{cases} a \cdot w_i^t & \text{if } \hat{w}_i^t \neq 0 \\ b + w_i^t & \text{if } \hat{w}_i^t = 0, w_i^t > 0 \\ b & \text{otherwise} \end{cases}, \quad (6)$$

$$w_{2,i}^b = \begin{cases} (1-a) w_i^t & \text{if } \hat{w}_i^t \neq 0 \\ -b & \text{if } \hat{w}_i^t = 0, w_i^t > 0 \\ -b + w_i^t & \text{otherwise} \end{cases}, \quad (7)$$

where a and b are the variables to solve. By Equations (6) and (7) with $\hat{\mathbf{w}}^t = \hat{\mathbf{w}}_1^b + \hat{\mathbf{w}}_2^b$, we get

$$\begin{aligned} a &= \frac{\sum_{i \in \mathcal{I}} |w_i^t| + \sum_{j \in \mathcal{J}} |w_j^t| - \sum_{k \in \mathcal{K}} |w_k^t|}{2 \sum_{i \in \mathcal{I}} |w_i^t|}, \\ b &= \frac{\frac{n}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} |w_i^t| - \sum_{i=1}^n |w_i^t|}{2(|\mathcal{J}| + |\mathcal{K}|)}, \end{aligned} \quad (8)$$

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 (8) is in Appendix A.

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). For word embedding, we use row-wise ternarization (i.e., one scaling parameter for each row in the embedding). After splitting, each of the two split matrices has its own scaling factor.

Aside from weight binarization, we simultaneously quantize activations before all matrix multiplications, which could accelerate inference on specialized hardwares (Shen et al., 2020; Zafrir et al., 2019). 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 calculations are negligible compared to matrix multiplication. The last classification layer is also not quantized to avoid a large accuracy drop.

Training with Knowledge Distillation. Knowledge distillation is shown to benefit BERT quantization (Zhang et al., 2020). Following (Jiao et al., 2020; Zhang et al., 2020), we first perform *intermediate-layer distillation* from the full-precision teacher network's embedding $\mathbf{E}$, layer-wise MHA output $\mathbf{M}_l$ and FFN output $\mathbf{F}_l$ to the quantized student counterpart $\hat{\mathbf{E}}, \hat{\mathbf{M}}_l, \hat{\mathbf{F}}_l$ ($l = 1, 2, ...L$). We aim to minimize their mean squared errors, i.e., $\ell_{emb} = \text{MSE}(\hat{\mathbf{E}}, \mathbf{E})$, $\ell_{mha} = \sum_l \text{MSE}(\hat{\mathbf{M}}_l, \mathbf{M}_l)$, and $\ell_{ffn} = \sum_l \text{MSE}(\hat{\mathbf{F}}_l, \mathbf{F}_l)$. Thus the objective function is

$$\ell_{int} = \ell_{emb} + \ell_{mha} + \ell_{ffn}. \tag{9}$$

We then conduct *prediction-layer distillation* by minimizing the soft cross-entropy (SCE) between quantized student logits $\hat{\mathbf{y}}$ and teacher logits $\mathbf{y}$, i.e.,

$$\ell_{pred} = \text{SCE}(\hat{\mathbf{y}}, \mathbf{y}). \tag{10}$$

Further Fine-tuning. After splitting from the half-sized ternary model, the binary model inherits its performance on a new architecture with full width. However, the original minimum of the ternary model may not hold in this new loss landscape after splitting. Thus we further fine-tune with prediction-layer distillation to look for a better solution. We dub the resulting model as BinaryBERT.

3.2 Adaptive Splitting

Our proposed approach also supports *adaptive splitting* that can flexibly adjust the width of BinaryBERT, based on the parameter sensitivity to binarization and resource constraints of edge devices.

Specifically, given the resource constraints $\mathcal{C}$ (e.g., model size and computational FLOPs), we first train a mixed-precision model adaptively (with sensitive parts being ternary and the rest being binary), and then split ternary weights into binary ones. Therefore, adaptive splitting finally enjoys consistent arithmetic precision (1-bit) for all weight matrices, which is usually easier to deploy than the mixed-precision counterpart.

Formulation. Intuitively, we assign ternary values to weight matrices that are more sensitive to quantization. The quantization sensitivity of the weight matrix is empirically measured by the performance gain of not quantizing it comparing to the fully-quantized counterpart (Details are in Appendix B.1.). We denote $\mathbf{u} \in \mathbb{R}_+^Z$ as the sensitivity vector, where Z is the total number of splittable weight matrices in all Transformer layers, the word embedding layer and the pooler layer. The cost vector $\mathbf{c} \in \mathbb{R}_+^Z$ stores the additional increase of parameter or FLOPs of each ternary weight matrix against a binary choice. The splitting assignment can be represented as a binary vector $\mathbf{s} \in \{0, 1\}^Z$, where $s_z = 1$ means to ternarize the z-th weight matrix, and vice versa. The optimal assignment $\mathbf{s}^*$ can thus be solved from the following combinatorial optimization problem:

$$\begin{aligned} \max_{\mathbf{s}} \quad & \mathbf{u}^\top \mathbf{s} \\ \text{s.t.} \quad & \mathbf{c}^\top \mathbf{s} \le \mathcal{C} - \mathcal{C}_0, \ \ \mathbf{s} \in \{0, 1\}^Z, \end{aligned} \tag{11}$$

where $\mathcal{C}_0$ is the baseline efficiency of the half-sized binary network. Dynamic programming can be applied to solve Equation (11) to avoid NP-hardness.

4 Experiments

In this section, we empirically verify our proposed approach on the GLUE (Wang et al., 2018) and SQuAD (Rajpurkar et al., 2016, 2018) benchmarks. 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 with other state-of-the-arts in Section 4.3, and finally provide more discussions on the proposed methods in Section 4.4. Code is available at https://github.com/huawei-noah/Pretrained-Language-Model/tree/master/BinaryBERT.

4.1 Experimental Setup

Dataset and Metrics. The GLUE benchmark contains multiple natural language understanding tasks. We follow Devlin et al. (2019) to evaluate the performance on these tasks: Matthews correlation

#	Quant	#Bits (W-E-A)	Size (MB)	FLOPs (G)	DA	MNLI -m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
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	83.9
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	80.8
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**	**82.7**
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**	78.4
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**	**79.9**
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	82.5
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**	**83.3**
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	81.9
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**	**82.6**

Table 1: Results on the GLUE development set. "#Bits (W-E-A)" represents the bit number for weights of Transformer layers, word embedding, and activations. "DA" is short for data augmentation. "Avg." denotes the average results of all tasks including MNLI-m and MNLI-mm. The higher results in each block are bolded.

#	Quant	#Bits (W-E-A)	Size (MB)	FLOPs (G)	DA	MNLI -m/mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Avg.
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	82.2
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	78.5
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**	**80.6**
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	76.3
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**	**78.5**
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	80.1
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**	**80.6**
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	79.2
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**	**80.2**

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

for CoLA, Spearman correlation for STS-B and accuracy for the rest tasks: RTE, MRPC, SST-2, QQP, 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 at inference. For quantized operations, we follow (Zhou et al., 2016; Liu et al., 2018; Li et al., 2020a) to count the bit-wise operations, i.e., the multiplication between 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. Following (Jiao et al., 2020; Hou et al., 2020; Zhang et al., 2020), we adopt data augmentation with one training epoch in each stage on all GLUE tasks except for MNLI and QQP. Aside from this default setting, we also remove data augmentation and perform vanilla training with 6 epochs on these tasks. On MNLI and QQP, we train 3 epochs for each stage.

We verify our ternary weight splitting (**TWS**) against vanilla binary training (**BWN**), the latter of which doubles training epochs to match the overall training time in TWS for fair comparison. More training details are provided in Appendix B.

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 8-bit uniform quantization (Zhang et al., 2020; Shen et al., 2020) in past efforts, we further pioneer to study 4-bit activation quantization. We find that uniform quantization can hardly deal with outliers in the activation. Thus we use Learned Step-size Quantization (LSQ) (Esser et al., 2019) to directly learn the quantized values, which empirically achieves better quantization performance.

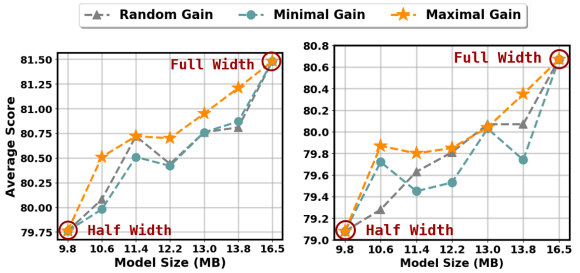
4.2 Experimental Results

4.2.1 Results on the GLUE Benchmark

The main results on the development set are shown in Table 1. For results without data augmenta-

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

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



(a) 8-bit Activation. (b) 4-bit Activation.

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

tion (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% ↑ and 9.6% ↑ 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 with 8-bit activations (row 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 with BWN.

4.2.2 Results on SQuAD Benchmark

The results on the development set of SQuAD v1.1 and v2.0 are shown in Table 3. 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 enjoys a larger gain in performance from the splitting approach. 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.

[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.

Method	#Bits (W-E-A)	Size (MB)	Ratio (↓)	SQuAD v1.1	MNLI -m
BERT-base	*full-prec.*	418	1.0	80.8/88.5	84.6
DistilBERT	*full-prec.*	250	1.7	79.1/86.9	81.6
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	79.7/87.5	82.8
ALBERT-E128	*full-prec.*	45	9.3	82.3/89.3	81.6
ALBERT-E768	*full-prec.*	120	3.5	81.5/88.6	82.0
Quant-Noise	PQ	38	11.0	-	83.6
Q-BERT	2/4-8-8	53	7.9	79.9/87.5	83.5
Q-BERT	2/3-8-8	46	9.1	79.3/87.0	81.8
Q-BERT	2-8-8	28	15.0	69.7/79.6	76.6
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	79.9/87.4	83.5
BinaryBERT	**1-1-8**	**17**	**24.6**	**80.8/88.3**	**84.2**
BinaryBERT	**1-1-4**	**17**	**24.6**	**79.3/87.2**	**83.9**

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

4.2.3 Adaptive Splitting

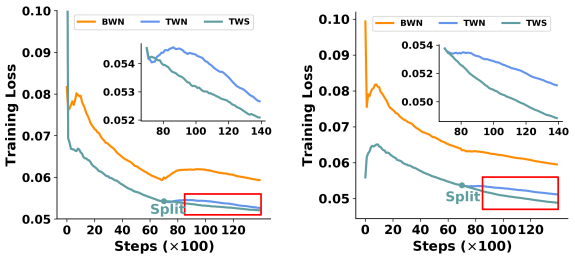
The adaptive splitting in Section 3.2 supports the conversion of mixed ternary and binary precisions for more-fine-grained configurations. To verify its advantages, we name our approach as Maximal Gain according to Equation (11), and compare it with two baseline strategies i) Random Gain that randomly 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. The end-points of 9.8MB and 16.5MB are the half-sized and full-sized BinaryBERT, respectively. As can be seen, adaptive splitting generally outperforms the other two baselines under varying model size, indicating the effectiveness of maximizing the gain in adaptive splitting. In Appendix C.4, we provide detailed performance on the six tasks, together with the architecture visualization of adaptive splitting.

4.3 Comparison with State-of-the-arts

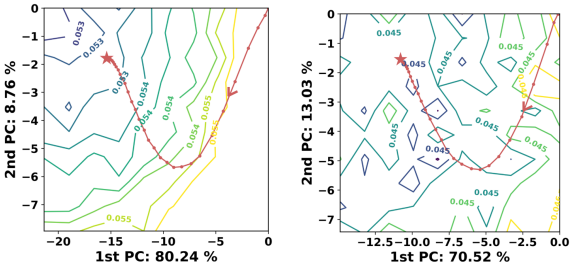
Now we compare our proposed approach with a variety of state-of-the-art counterparts, including Q-BERT (Shen et al., 2020), GOBO (Zadeh and Moshovos, 2020), Quant-Noise (Fan et al., 2020) and TernaryBERT (Zhang et al., 2020). Aside from quantization, we also compare with other general compression approaches such as DistillBERT (Sanh et al., 2019), LayerDrop (Fan et al., 2019), TinyBERT (Jiao et al., 2020), and 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 quantiza-

Quant	#Bits (W-E-A)	SQuAD v1.1	MNLI -m	QNLI	MRPC
$\text{TWN}_{0.5\times}$	2-2-8	80.3/87.9	84.1	91.3	85.7
$\text{TWS}_{1.0\times}$	1-1-8	**80.8/88.3**	**84.2**	**91.6**	**86.0**
$\text{TWN}_{0.5\times}$	2-2-4	78.0/86.4	83.7	90.9	85.5
$\text{TWS}_{1.0\times}$	1-1-4	**79.3/87.2**	**83.9**	**91.4**	**86.0**

Table 5: The performance gain by fine-tuning the binary model after splitting. $0.5\times$ and $1.0\times$ denote the half-sized and full-sized models, respectively.



(a) 8-bit Activation. (b) 4-bit Activation.



(c) 8-bit Activation. (d) 4-bit Activation.

Figure 6: (a) and (b) show the training curves on MRPC under different activation bits. The red box is enlarged in the sub-figure. (c) and (d) visualize the fine-tuning trajectories after splitting, on the 2-D loss contour of BinaryBERT.

tion approaches. Compared with the full-precision model, our BinaryBERT retains competitive performance with a significant reduction of model size and computation. For example, we achieve more than $\mathbf{24}\times$ compression ratio compared with BERT-base, with only $0.4\% \downarrow$ and $0.0\%/0.2\% \downarrow$ drop on MNLI-m on SQuAD v1.1, respectively.

4.4 Discussion

4.4.1 Further Improvement after Splitting

We now demonstrate the performance gain by refining the binary model on the new architecture. We evaluate the performance gain after splitting from a half-width ternary model ($\text{TWN}_{0.5\times}$) to the full-sized model ($\text{TWN}_{1.0\times}$) on the development set of SQuAD v1.1, MNLI-m, QNLI 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.

Quant	#Bits (W-E-A)	SQuAD v1.1	MNLI -m	QNLI	SST-2
BWN	1-1-8	79.2/86.9	**84.2**	91.2	92.7
LAB	1-1-8	79.0/87.0	83.6	91.5	92.8
BiReal	1-1-8	79.4/87.1	83.9	91.4	92.5
BWN†	1-1-8	79.4/87.3	**84.2**	91.3	92.8
BWN‡	1-1-8	79.6/87.2	83.5	91.2	92.9
TWS	1-1-8	**80.8/88.3**	**84.2**	**91.6**	**93.2**
BWN	1-1-4	77.5/85.8	83.5	91.2	92.5
LAB	1-1-4	76.7/85.5	83.3	91.3	92.9
BiReal	1-1-4	76.9/85.4	83.4	91.0	92.8
BWN†	1-1-4	78.2/86.2	83.6	91.3	92.9
BWN‡	1-1-4	78.3/86.5	83.1	90.9	92.9
TWS	1-1-4	**79.3/87.2**	**83.9**	**91.4**	**93.7**

Table 6: Comparison with other binarization methods.

Training Curves. Furthermore, we plot the training loss curves of BWN, TWN and our TWS on MRPC with data augmentation in Figures 6(a) and 6(b). Since TWS cannot inherit the previous optimizer due to the architecture change, we reset the optimizer and learning rate scheduler of BWN, TWN and TWS for a fair comparison, despite the slight increase of loss after splitting. We find that our TWS attains much lower training loss than BWN, and also surpasses TWN, verifying the advantages of fine-tuning on the wider architecture.

Optimization Trajectory. We also follow (Li et al., 2018; Hao et al., 2019) to visualize the optimization trajectory after splitting in Figures 6(c) and 6(d). We calculate the first two principal components of parameters in the final BinaryBERT, which are the basis for the 2-D plane. The loss contour is thus obtained by evaluating each grid point in the plane. It is found that the binary models are heading towards the optimal solution for both 8/4-bit activation quantization on the loss contour.

4.4.2 Exploring More Binarization Methods

We now study if there are any improved binarization variants that can directly bring better performance. Aside from BWN, we compare with LAB (Hou et al., 2017) and BiReal (Liu et al., 2018). Meanwhile, we compare with gradual quantization, i.e., BWN training based on a ternary model, denoted as BWN†. Furthermore, we also try the same scaling factor of BWN with TWN to make the precision change smooth, dubbed as BWN‡. From Table 6, we find that our TWS still outperforms various binarization approaches in most cases, suggesting the superiority of splitting in finding better minima than direct binary training.

5 Related Work

Network quantization has been a popular topic with vast literature in efficient deep learning. Below we give a brief overview for three research strands: network binarization, mixed-precision quantization and neuron splitting, all of which are related to our proposed approach.

5.1 Network Binarization

Network binarization achieves remarkable size reduction and is widely explored in computer vision. Existing binarization approaches can be categorized into quantization error minimization (Rastegari et al., 2016; Hou et al., 2017; Zhang et al., 2018), improving training objectives (Martinez et al., 2020; Bai et al., 2020) and reduction of gradient mismatch (Bai et al., 2018; Liu et al., 2018, 2020). Despite the empirical success of these approaches in computer vision, there is little exploration of binarization in natural language processing tasks. Previous works on BERT quantization (Zafrir et al., 2019; Shen et al., 2020; Zhang et al., 2020) push down the bit-width to as low as two, but none of them achieves binarization. On the other hand, our work serves as the first attempt to binarize the pre-trained language models.

5.2 Mixed-precision Quantization

Given the observation that neural network layers exhibit different sensitivity to quantization (Dong et al., 2019; Wang et al., 2019), mixed-precision quantization re-allocate layer-wise quantization bit-width for higher compression ratio. Inspired by neural architecture search (Liu et al., 2019; Wang et al., 2020), common approaches of mixed-precision quantization are primarily based on differentiable search (Wu et al., 2018a; Li et al., 2020b), reinforcement learning (Wu et al., 2018b; Wang et al., 2019), or simply loss curvatures (Dong et al., 2019; Shen et al., 2020). While mixed-precision quantized models usually demonstrate better performance than traditional methods under the same compression ratio, they are also harder to deploy (Habi et al., 2020). On the contrary, BinaryBERT with adaptive splitting enjoy both the good performance from the mixed precision of ternary and binary values, and the easy deployment given the consistent arithmetic precision.

There are also works on binary neural architecture search (Kim et al., 2020; Bulat et al., 2020) which have a similar purpose to mixed-precision quantization. Nonetheless, such methods are usually time-consuming to train and are prohibitive for large pre-trained language models.

5.3 Neuron Splitting

Neuron splitting is originally proposed to accelerate the network training, by progressively increasing the width of a network (Chen et al., 2016; Wu et al., 2019). The split network equivalently inherits the knowledge from the antecessors and is trained for further improvement. Recently, neuron splitting is also studied in quantization (Zhao et al., 2019; Kim et al., 2019). By splitting neurons with large magnitudes, the full-precision outliers are removed and thus the quantization error can be effectively reduced (Zhao et al., 2019). Kim et al. (2019) apply neuron splitting to decompose ternary activation into two binary activations based on bias shifting of the batch normalization layer. However, such a method cannot be applied in BERT as there is no batch normalization layer. Besides, weight splitting is much more complex due to the equivalence constraint on both the quantized and latent full-precision weights.

6 Conclusion

In this paper, we propose BinaryBERT, pushing 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 thus propose a ternary weight splitting that splits a trained ternary BERT to initialize BinaryBERT, followed by fine-tuning for further refinement. Our approach also supports adaptive splitting that can tailor the size of BinaryBERT based on the edge device constraints. Empirical results show that our approach significantly outperforms vanilla binary training, achieving state-of-the-art performance on BERT compression.

Acknowledgement

This work was partially supported by the National Key Research and Development Program of China (No. 2018AAA0100204), and Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14210717 of the General Research Fund). We sincerely thank all anonymous reviewers for their insightful suggestions.

References

H. Bai, J. Wu, I. King, and M. Lyu. 2020. Few shot network compression via cross distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 3203–3210.

Y. Bai, Y. Wang, and E. Liberty. 2018. Proxquant: Quantized neural networks via proximal operators. In *International Conference on Machine Learning*.

A. Bulat, B. Martinez, and G. Tzimiropoulos. 2020. Bats: Binary architecture search. In *European Conference on Computer Vision*, pages 309–325.

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*.

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*.

Z. Dong, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 293–302.

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.

H. Habi, R. Jennings, and A. Netzer. 2020. Hmq: Hardware friendly mixed precision quantization block for cnns. In *European Conference on Computer Vision*, pages 448–463.

Y. Hao, L. Dong, F. Wei, and K. Xu. 2019. Visualizing and understanding the effectiveness of BERT. In *Conference on Empirical Methods in Natural Language Processing*.

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*.

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

L. Hou, Yao Q., and J. T. Kwok. 2017. Loss-aware binarization of deep networks. In *International Conference on Learning Representations*.

Z. Huang, L Hou, L. Shang, X. Jiang, X. Chen, and Q. Liu. 2021. Ghostbert: Generate more features with cheap operations for bert. In *Annual Meeting of the Association for Computational Linguistics*.

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

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*.

D. Kim, K Singh, and J. Choi. 2020. Learning architectures for binary networks. In *European Conference on Computer Vision*, pages 575–591.

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.

H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. 2018. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*.

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

Y. Li, W. Wang, H. Bai, R. Gong, X. Dong, and F. Yu. 2020b. Efficient bitwidth search for practical mixed precision neural network. Preprint arXiv:2003.07577.

H. Liu, K. Simonyan, and Y. Yang. 2019. Darts: Differentiable architecture search. In *International Conference on Learning Representations*.

Z. Liu, Z. Shen, M. Savvides, and K. Cheng. 2020. Reactnet: Towards precise binary neural network with generalized activation functions. In *European Conference on Computer Vision*, pages 143–159.

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*.

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*.

B. Martinez, J. Yang, A. Bulat, and G. Tzimiropoulos. 2020. Training binary neural networks with real-to-binary convolutions. In *International Conference on Learning Representations*.

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

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*.

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 *Proceedings of the 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*.

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*.

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.

J. Wang, H. Bai, J. Wu, X. Shi, J. Huang, I. King, M. Lyu, and J. Cheng. 2020. Revisiting parameter sharing for automatic neural channel number search. In *Advances in Neural Information Processing Systems*, volume 33.

K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8612–8620.

B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer. 2018a. Mixed precision quantization of convnets via differentiable neural architecture search. Preprint arXiv:1812.00090.

J. Wu, Y. Zhang, H. Bai, H. Zhong, J. Hou, W. Liu, and J Huang. 2018b. Pocketflow: An automated framework for compressing and accelerating deep neural networks. In *Advances in Neural Information Processing Systems, Workshop on Compact Deep Neural Networks with Industrial Applications*.

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

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.

D. Zhang, J. Yang, D. Ye, and G. Hua. 2018. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *European conference on computer vision*, pages 365–382.

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

R. Zhao, Y. Hu, J. Dotzel, C. De Sa, and Z. Zhang. 2019. Improving neural network quantization without retraining using outlier channel splitting. In *International Conference on Machine Learning*.

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 (8)

In this section, we show the derivations to obtain a and b. Recall the BWN quantizer introduced in Section 2, we have

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

where

$$\alpha_1 = \frac{1}{n}\big[\sum_{i\in\mathcal{I}}|aw^t_i| + \sum_{i\in\mathcal{J}}|w^t_j + b| + \sum_{i\in\mathcal{K}}|b|\big].$$

Similarly,

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

where

$$\alpha_2 = \frac{1}{n}\big[\sum_{i\in\mathcal{I}}|(1-a)w^t_i| + \sum_{j\in\mathcal{J}}|-b| + \sum_{k\in\mathcal{K}}|w^t_k - b|\big].$$

According to $\hat{\mathbf{w}}^t = \hat{\mathbf{w}}^b_1 + \hat{\mathbf{w}}^b_2$, for those $\hat{w}^t_i = \hat{w}^b_{1,i} + \hat{w}^b_{2,i} = 0$, we have

$$\begin{aligned}
&\frac{1}{n}\big[\sum_{i\in\mathcal{I}}|aw^t_i| + \sum_{j\in\mathcal{J}}|w^t_j + b| + \sum_{k\in\mathcal{K}}|b|\big] \\
&= \frac{1}{n}\big[\sum_{i\in\mathcal{I}}|(1-a)w^t_i| + \sum_{j\in\mathcal{J}}|-b| + \sum_{k\in\mathcal{K}}|w^t_k - b|\big].
\end{aligned}$$

By assuming $0 < a < 1$ and $b > 0$, this can be further simplified to

$$a\sum_{i\in\mathcal{I}}|w^t_i| + \sum_{j\in\mathcal{J}}|w^t_j| = (1-a)\sum_{i\in\mathcal{I}}|w^t_i| + \sum_{k\in\mathcal{K}}|w^t_k|,$$

which gives the solution of a as

$$a = \frac{\sum_{i\in\mathcal{I}}|w^t_i| + \sum_{j\in\mathcal{J}}|w^t_j| - \sum_{k\in\mathcal{K}}|w^t_k|}{2\sum_{i\in\mathcal{I}}|w^t_i|}.$$

We empirically find the solution satisifies $0 < a < 1$. For $\hat{w}^t_i \neq 0$, from $\hat{w}^t_i = \hat{w}^b_{1,i} + \hat{w}^b_{2,i}$, we have

$$\begin{aligned}
&\frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}|w^t_i| = \alpha_1 + \alpha_2 \\
&= \frac{1}{n}\big[\sum_{i\in\mathcal{I}}|aw^t_i| + \sum_{j\in\mathcal{J}}|w^t_j + b| + \sum_{k\in\mathcal{K}}|b|\big] \\
&+ \frac{1}{n}\big[\sum_{i\in\mathcal{I}}|(1-a)w^t_i| + \sum_{j\in\mathcal{J}}|-b| + \sum_{k\in\mathcal{K}}|w^t_k - b|\big] \\
&= \frac{1}{n}\big[\sum_{i\in\mathcal{I}}|w^t_i| + \sum_{j\in\mathcal{J}}|w^t_j| + \sum_{k\in\mathcal{K}}|w^t_k| \\
&+ 2\sum_{j\in\mathcal{J}}|b| + 2\sum_{k\in\mathcal{K}}|b|\big] \\
&= \frac{1}{n}\big[\sum_{i=1}^{n}|w^t_i| + 2(|\mathcal{J}| + |\mathcal{K}|)\cdot b\big].
\end{aligned}$$

Thus the solution for b is

$$b = \frac{\frac{n}{|\mathcal{I}|}\sum_{i\in\mathcal{I}}|w^t_i| - \sum_{i=1}^{n}|w^t_i|}{2(|\mathcal{J}| + |\mathcal{K}|)},$$

which satisfies $b > 0$.

B Implementation Details

B.1 Detailed Procedure of Adaptive Splitting

As mentioned in Section 3.2, the adaptive splitting requires to first estimate the quantization sensitivity vector $\mathbf{u}$. We study the sensitivity in two aspects: the Transformer parts, and the Transformer layers. For Transformer parts, we follow the weight categorization in Section 2.2: MHA-Q/K, MHA-V, MHA-O, FFN-Mid and FFN-Out. For each of them, we compare the performance gap between quantizing and not quantizing that part (e.g., MHA-V), while leavging the rest parts all quantized (e.g., MHA-Q/K, MHA-O, FFN-Mid and FFN-Out). Similarly, for each Transformer layer, we quantize all layers but leave the layer under investigation unquantized, and calculate the performance gain compared with the fully qauntized baseline. The performance gain of both Transformer parts and layers are shown in Figure 7. 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.

However, the absolute performance gain may not reflect the quantization sensitivity directly, since Transformer parts have different number of parameters. Therefore, we divide the performance gain by the number of parameters in that part or layer to obtain the parameter-wise performance gain. We are thus able to measure the quantization sensitivity of the ith Transformer part in the jth Transformer layer by summing their parameter-wise performance gain together. We also apply the same procedure to word embedding and pooler layer to otain their sensitivity scores.

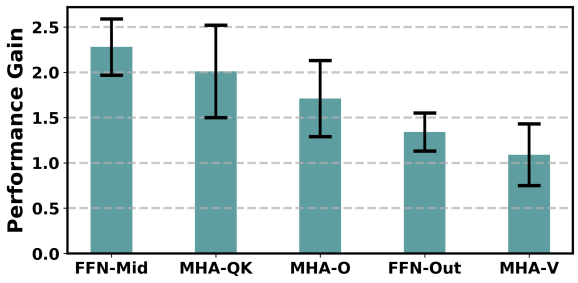
We are now able to solve Equation (11) by dynamic programming. The combinatorial optimization can be viewed as a knapsack problem, where the constraint $\mathcal{C} - \mathcal{C}_0$ is the volume of the knapsack, and the sensitivity scores $\mathbf{u}$ are the item values.

B.2 Hyper-parameter Settings

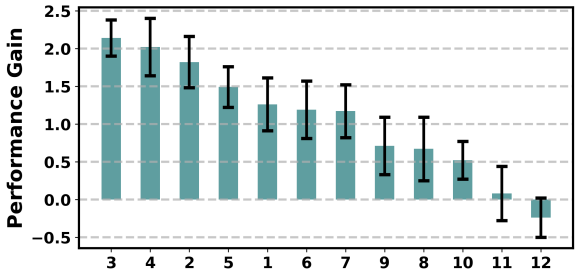
We first perform the two-stage knowledge distillation, i.e., intermediate-layer distillation (Int. Dstil.) and prediction-layer distillation (Pred. Dstil.) on

	BinaryBERT		
	Int. Dstil. (Ternary)	Pred. Dstil. (Ternary)	Split Ft. (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 datasets	6	6	6
Epochs w DA -other datasets	1	1	1
Epochs w/o DA -MNLI, QQP	3	3	3

Table 7: Hyper-parameters for training BinaryBERT on the GLUE benchmark at different stages.



(a) Transformer Parts.



(b) Transformer Layers.

Figure 7: The performance gain of different Transformer parts and layers in descending order. All numbers are averaged by 10 random runs with standard deviations reported.

the ternary model, and then perform ternary weight splitting followed by fine-tuning (Split Ft.) 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 7.

C More Empirical Results

C.1 Performance Drop by Binarization

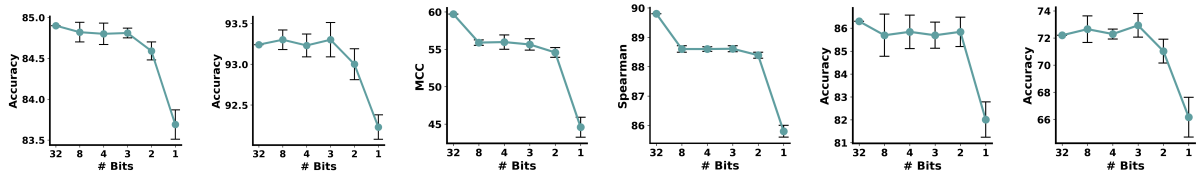
Here we provide more empirical results on the sharp drop in performance as a result of binarization. We run multi-bit quantization on the BERT model over representative tasks of the GLUE benchmark, and activations are quantized in both 8-bit and 4-bit. We run 10 independent experiments for each task except for MNLI with 3 runs. We follow the same procedure in Section 2.1, and the default experimental setup in Appendix B.2 without data augmentation and splitting. The results are shown in Figures 8 and 9 respectively. It can be found that while the performance drops slowly from full-precision to ternarization, there is a consistent sharp drop by binarization in each tasks and on both 8-bit and 4-bit activation quantization. This is similar to the findings in Figure 1.

C.2 More Visualizations of Loss Landscape

To comprehensively compare the loss curvature among the full-precision, ternary and binary models, we provide more landscape visualizations aside from the value layer in Figure 2. We extract parameters from MHA-K, MHA-O, FFN-Mid and FFN-out in the first two Transformer layers, and the corresponding landscape are shown in Figure 10, Figure 11, Figure 12, Figure 13 respectively. We omit MHA-Q due to page limitation, and also it is symmetric to MHA-K with similar landscape observation. It can be found that binary model have steep and irregular loss landscape in general w.r.t different parameters of the model, and is thus hard to optimize directly.

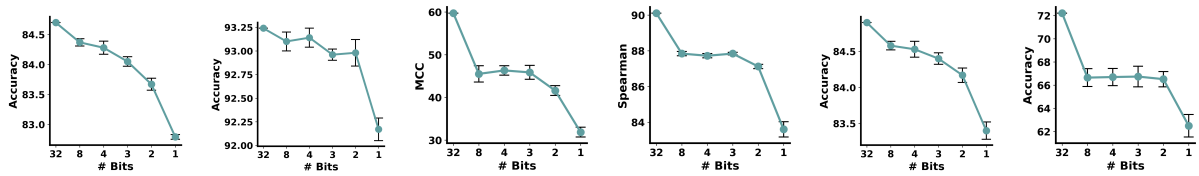
C.3 Ablation of Knowledge Distillation

While knowledge distillation on BERT has been thoroughly investigated in (Jiao et al., 2020; Hou et al., 2020; Zhang et al., 2020), here we further conduct ablation study of knowledge distillation on the proposed ternary weight splitting. We compare with no distillation ("N/A"), prediction distillation ("Pred") and our default setting ("Int.+Pred"). For "N/A" or "Pred", fine-tuning after splitting follows the same setting to their ternary



(a) MNLI-m. (b) SST-2. (c) CoLA. (d) STS-B. (e) MRPC. (f) RTE.

Figure 8: Performance of quantized BERT with different weight bits and 8-bit activation on the GLUE Benchmarks. The results are obtained from 10 random seeds except for MNLI with 3 seeds.



(a) MNLI-m. (b) SST-2. (c) CoLA. (d) STS-B. (e) MRPC. (f) RTE.

Figure 9: Performance of quantized BERT with different weight bits and 4-bit activation on the GLUE Benchmarks. The results are obtained from 10 random seeds except for MNLI with 3 seeds.

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

Table 8: 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.
10.6	Min.	90.6	92.6	51.7	87.4	85.3	70.8	79.7
	Rand.	91.1	92.7	51.3	87.6	84.8	68.2	79.3
	Max.	90.9	92.7	53.5	87.5	84.6	70.0	79.9
11.4	Min.	90.9	92.8	50.9	87.6	85.3	69.4	79.5
	Rand.	90.8	92.8	51.7	87.5	84.6	70.4	79.6
	Max.	91.1	92.6	52.1	87.7	85.3	70.0	79.8
12.2	Min.	90.9	92.7	50.8	87.6	84.8	70.4	79.5
	Rand.	91.2	93.0	52.0	87.6	85.1	70.0	79.8
	Max.	90.9	92.9	52.2	87.6	85.1	70.4	79.9
13.0	Min.	91.1	92.8	52.6	87.7	86.3	69.7	80.0
	Rand.	91.3	93.0	52.9	87.8	85.8	69.7	80.1
	Max.	91.3	92.9	53.4	87.8	85.3	69.7	80.1
13.8	Min.	91.1	93.1	51.5	87.9	84.8	70.0	79.7
	Rand.	91.3	92.9	52.3	87.7	85.1	71.1	80.1
	Max.	91.3	92.8	53.6	88.0	85.8	70.8	80.4

Table 9: Results on GLUE development set for adaptive splitting with 4-bit activation quantization.

training. "Int.+Pred" follows our default setting in Table . We do not adopt data-augmentation, and results are shown in Table 10. It can be found that "Int.+Pred." outperforms both "N/A" and "Pred." with a clear margin, which is consistent to the findings in (Zhang et al., 2020) that knowledge distillation helps BERT quantization.

KD	#Bits (W-E-A)	MNLI (-m)	SST-2	CoLA	MRPC
N/A	1-1-8	83.2	92.1	49.2	82.8
Pred.	1-1-8	84.0	91.7	48.6	84.1
Int.+Pred.	1-1-8	**84.2**	**92.6**	**53.4**	**85.5**
N/A	1-1-4	82.6	90.9	39.2	76.5
Pred.	1-1-4	83.4	**92.3**	38.9	76.2
Int.+Pred.	1-1-4	**83.9**	**92.3**	**44.4**	**83.3**

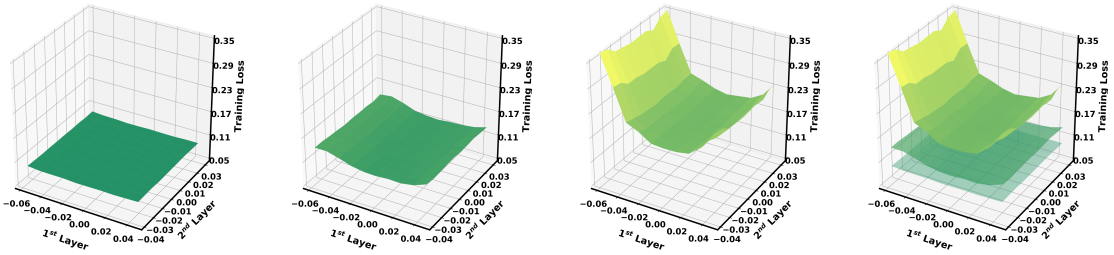
Table 10: Ablation study on knowledge distillation.

C.4 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 8 and Table 9. 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.

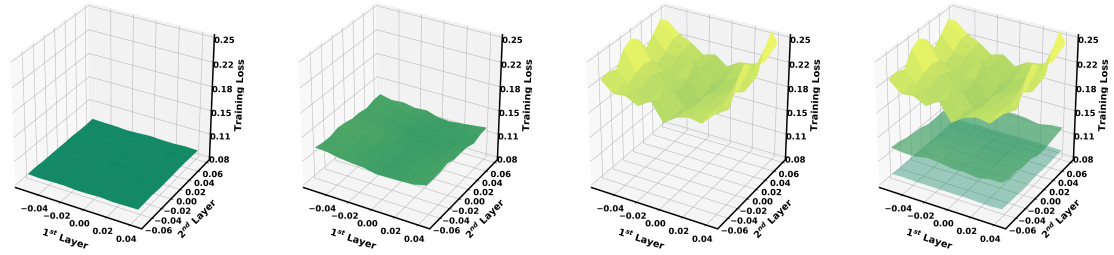
C.5 Architecture Visualization

We further visualize the architectures after adaptive splitting on MRPC in Figure 14. 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 14, with the increasing model size, shallower layers are more preferred for splitting than deeper layers, which is consistent to the findings in Figure 7.



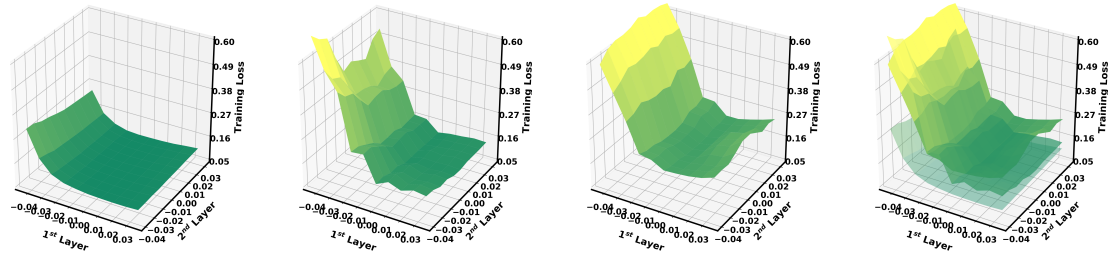
(a) Full-precision Model. (b) Ternary Model. (c) Binary Model. (d) All Together.

Figure 10: Loss landscape visualizations w.r.t MHA-K parameters of the 1st and 2nd Transformer layers on MRPC.



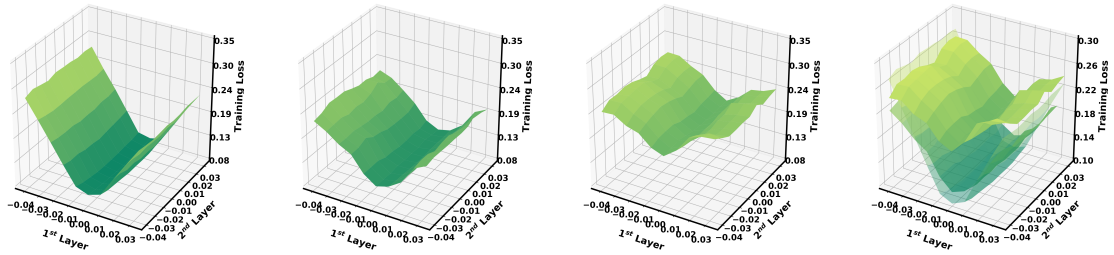
(a) Full-precision Model. (b) Ternary Model. (c) Binary Model. (d) All Together.

Figure 11: Loss landscape visualizations w.r.t MHA-Out parameters of the 1st and 2nd Transformer layers on MRPC.



(a) Full-precision Model. (b) Ternary Model. (c) Binary Model. (d) All Together.

Figure 12: Loss landscape visualizations w.r.t FFN-Mid parameters of the 1st and 2nd Transformer layers on MRPC.



(a) Full-precision Model. (b) Ternary Model. (c) Binary Model. (d) All Together.

Figure 13: Loss landscape visualizations w.r.t FFN-Out parameters of the 1st and 2nd Transformer layers on MRPC.

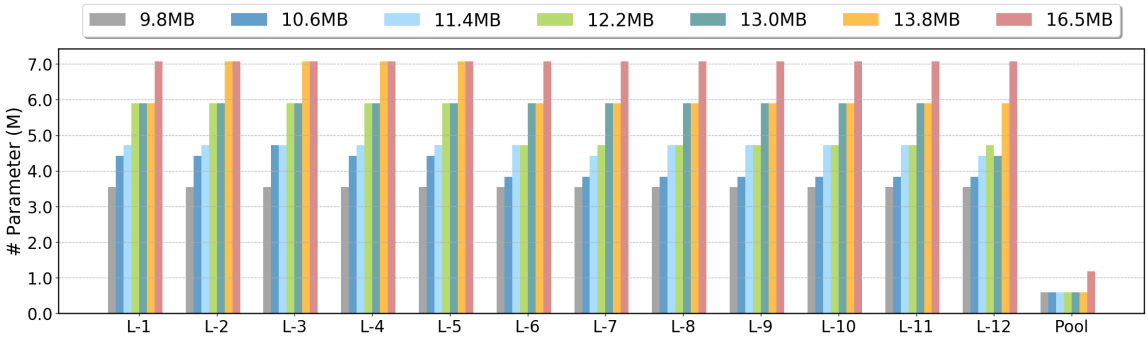


Figure 14: The architecture visualization for adaptive splitting on MRPC. The y-axis records the number of parameters split in each layer instead of the storage.