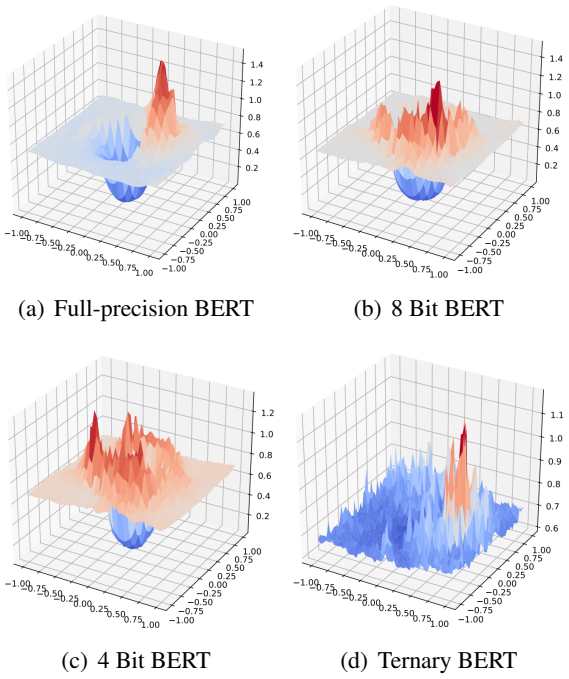
Efficient Two-Stage Progressive Quantization of BERT

Phuoc-Hoan Charles Le, Arash Ardakani, Amir Ardakani, Hang Zhang, Yuyan Chen, James J. Clark, Brett H. Meyer, Warren J. Gross
McGill University
Montreal, Canada

Abstract

The success of large BERT models has raised the demand for model compression methods to reduce model size and computational cost. Quantization can reduce the model size and inference latency, making inference more efficient, without changing its stucture, but it comes at the cost of performance degradation. Due to the complex loss landscape of ternarized/binarized BERT, we present an efficient two-stage progressive quantization method in which we fine tune the model with quantized weights and progressively lower its bitwidth, and then we fine tune the model with quantized weights and activations. At the same time, we strategically choose which bitwidth to fine-tune on and to initialize from, and which bitwidth to fine-tune under augmented data to outperform the existing BERT binarization methods without adding an extra module, compressing the binary model 18% more than previous binarization methods or compressing BERT by 31x w.r.t. to the full-precision model. Without data augmentation, we can outperform existing BERT ternarization methods.



(a) Full-precision BERT (b) 8 Bit BERT

(c) 4 Bit BERT (d) Ternary BERT

Figure 1: Loss landscape of the BERT model on the MRPC dataset with different weight bits.

1 Introduction

BERT (Devlin et al., 2019) models have demonstrated remarkable performance on NLP tasks. However, their memory and high computational cost make it difficult to fit them onto edge devices with limited resources for inference.

Recently, a number of methods have been developed to reduce the number of trainable parameters of BERT such as (Sun et al., 2020; Jiao et al., 2020). However, some edge devices require low-precision models for deployment due to the structure of their arithmetic units (Cortex-M, 2020). Therefore, quantization of BERT-based models is needed to avoid/minimize costly floating-point operations.

Previous works have tried to quantize BERT models. TernaryBERT (Zhang et al., 2020) used knowledge distillation to transfer the knowledge of the full-precision BERT to a BERT model with weights that are ternarized into $\{-1, 0, 1\}$ using 2 bits. BinaryBERT (Bai et al., 2021) binarized the weights into $\{-1, 1\}$ using 1 bit by exploiting the adaptive width property of DynaBERT (Hou et al., 2020) using a method called ternary weight splitting. BiBERT (Qin et al., 2022) managed to outperform (Bai et al., 2021) when binarizing acitvations and binarizing weights. However, there is always a performance loss associated with all the aforementioned methods when using ternary weights and to keep competitive performance, data augmentation from (Jiao et al., 2020) is needed, requiring up to 60x more data as shown in Table 8. For binary weights, the architecture is also modified and an extra module is added and with BiBERT's method alone it cannot reach the full precision performance.

Proceedings of The Third Workshop on Simple and Efficient Natural Language Processing (SustaiNLP), pages 1 - 9

More details will be explained on Appendix E on the BiBERT on non binary activations and weights.

A binarized/ternarized BERT has a lot more complex landscape than a full-precision BERT (Bai et al., 2021) as shown in Figure 1, so training ternary/binary model from scratch could cause the model to get trapped in a poor local minima. Therefore, we use a two-stage progressive quantization method inspired by (Zhuang et al., 2018). In the first stage, we progressively lower the bitwidth of weights, and subsequently fine-tune the model. More specifically, the model with a larger bitwidth is used to initialize the model for fine-tuning on a smaller bitwidth of weights. After reaching the best performance with quantized weights (e.g., binary weights), the BERT model is then fine-tuned with quantized activations as the second stage of the quantization method. Activations are also progressively quantized until the desired quantization bitwidth in a similar fashion.

However, for example, applying the method from (Zhuang et al., 2018) to ternarize BERT, naively, costs too much time (or up to 1.67x longer than necessary) with data augmentation used for every bitwidth of weights, with no performance gain as shown in Table 3. We therefore propose an efficient two-stage progressive quantization (ETSPQ) method in which we choose which bitwidth to progressively fine-tune on and which bitwidth to fine-tune under augmented data to save training time and achieve the best possible performance.

We are the first to ternarize the weights of BERT, compressing it by 14.9x while outperforming the full-precision model performance across nearly all GLUE datasets and without data augmentation it can outperform existing BERT ternarization methods. Also, we outperform the state-of-the-art BERT binarization in performance without adding an extra module, compressing the model 18% more than (Bai et al., 2021).

2 Preliminaries

A quantized model has a set of full precision weights, $\mathbf{w}$. For the forward pass, a quantization function from (Li et al., 2016) is used to ternarize each element w_i in the weight matrix $\mathbf{w}$ into $\hat{w}_i^t$ by

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

where $\Delta = \frac{0.7}{n}\|\mathbf{w}\|_1$, $\alpha = \frac{1}{|I|}\sum_{i \in I}|w_i|$, and $I = \{i|w_i \neq 0\}$. α is a scaling factor that is multiplied by ternary values.

For binarization, w_i is binarized into $\hat{w}_i^b$ during inference using the binarization method from (Rastegari et al., 2016) by

$$\hat{w}_i^b = \alpha(\text{sign}(w_i)) \quad (2)$$

where $\alpha = \frac{1}{n}\|\mathbf{w}\|_1$. α is a scaling factor that is multiplied by binary values. The activations x are quantized into $\hat{x}$ using the quantization function

$$\hat{x} = \text{round}((x - x_{min})/s) \cdot s + x_{min}. \quad (3)$$

where $s = (x_{max} - x_{min})/(2^{bitwidth} - 1)$

For training, we calculate the gradient of the distillation loss from Eq. 16 w.r.t the quantized weights and update the full-precision weight of the quantized model. Since the quantization function in Eq 1, 2, and 3 are not differentiable, a straight-through estimator adopted by (Courbariaux et al., 2015) is used to back propagate through the quantization function 1, 2, and/or 3. With this, the gradient $\frac{\partial \hat{x}}{\partial x}$ is approximated as an identity, so $\frac{\partial L}{\partial x}$ can be calculated as

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} \quad (4)$$

In our work, we use the knowledge distillation method from (Zhang et al., 2020; Bai et al., 2021) during fine-tuning as shown in Eq. 16. Which parts of the BERT are quantized can be seen on Appendix A.

3 Efficient Two Stage Progressive Quantization Method

With the complex loss landscape of binary/ternary models, training with binary/ternary weights and with quantized activations directly could lead the BERT model to converge to a poor local minimum. Therefore, we use an efficient two-stage progressive quantization (ETSPQ) method. The details of the ETSPQ method are provided in Alg 1. In the first stage, we progressively lower the bitwidth of weights down to binary (or to our desired bitwidth) while fine-tuning it on the target downstream task. The first stage allows us to have a good starting point as we are going to use our binarized/ternarized model weights to then fine-tune the model with quantized activations. In the 2^{nd} stage, with our binarized/ternarized model weights,

Model	W-A	Size (MB)	FLOPs (G)	CoLA	SST2	MRPC	STS-B	QNLI	QQP	MNLI-m	RTE	avg
Full precision	32-32	417.6	22.5	59.9	93.6	87.3	89.6	91.8	91.4	84.7	72.9	83.9
TernaryBERT	2-8	28.0	6.4	55.7	92.8	87.5	87.7	91.5	90.1	83.5	72.2	82.6
ETSPQ w/o aug.	2-8	28.0	6.4	55.4	93.2	87.7	89.3	91.8	**91.4**	**84.8**	74.0	83.1
ETSPQ	2-8	28.0	6.4	**58.8**	**93.9**	**88.0**	**89.8**	**92.0**	**91.4**	**84.8**	**76.9**	**84.5**
BinaryBERT	1-8	16.5	3.1	55.5	93.2	86.0	89.2	**91.6**	91.2	**84.2**	74.0	83.1
ETSPQ	1-8	13.4	3.1	**55.6**	**93.9**	**88.2**	**89.6**	**91.6**	**91.3**	**84.2**	**74.7**	**83.6**
BinaryBERT	1-4	16.5	1.5	53.3	**93.7**	86.0	88.6	**91.4**	**91.2**	**83.9**	71.5	82.6
ETSPQ	1-4	13.4	1.5	**56.2**	93.0	**88.2**	**88.9**	90.6	91.0	83.6	**74.7**	**83.2**

Table 1: Quantization performance for GLUE dev dataset. W-A stands for the bit width of weights and activations.

we then progressively lower the activation bits.

However, to save training time at the same time getting the best possible performance, we choose which bitwidth to finetune on and to initialize from and which bitwidth to fine-tune under augmented data based on the performance of BERT under that bitwidth, unlike (Zhuang et al., 2018).

Algorithm 1: Two-stage progressive quantization

Input: Train/Dev data or Aug train data; 32-bit teacher model; Student model initialized from the teacher model

Output: student model with k-bit weights and p-bit activations

```
/* Progressively reduce
   bitwidth of weights      */
wbits = [32, 8, 2, 1]; abits = [32, 8, 4]
for k in range(1,4) do
    -Initialize the student model weights
     from a model with wbits[k-1] bit
     weights
    -Use augmented data if wbits[k] ≤ 2,
     else use non-augmented data
    for epoch=1,...,max epoch do
        for t=1,...,data size do
            train student and quantize
             student with wbits[k] bit
             weights and save the best
             student model
/* Progressively reduce
   bitwidth of activations  */
for p in range(1,3) do
    -Initialize the student model weights
     from a model with wbits[k] bit weights
     and abits[p-1] bit activations
    for epoch=1,...,max epoch do
        for t=1,...,data size do
            train the quantized student
             model with abits[p] bit
             activations and wbits[k] bit
             weights and save the best
             student model
```

3.1 Selective Bitwidth Finetuning

Rather than progressively quantizing weights/activations i.e., 32-bit→16-bit→8-bit→4-bit→2-bit→1-bit as in (Zhuang et al., 2018) for each stage, we progressively quantizes the weights, i.e., 32-bit→8-bit →2-bit→1-bit, in the first stage, and then the activations, i.e., 32-bit→8-bit→4-bit, in the second stage, as we found that for BERT, the performance doesn't change a lot if we fine-tune for more different bit-widths as shown in Table 2. From (Bai et al., 2021), the performance doesn't drop until weights are ternarized and there is no point in fine-tuning a 16-bit BERT model since an 8-bit BERT model could reach the same performance as the full precision.

Also, the performance barely changes when using a 4-bit BERT model as a starting point versus using a 8-bit BERT model as a starting point when finetuning a 2-bit BERT model, so we just use an 8-bit BERT model as a starting point.

3.2 Selective Data Augmentation

Data augmentation was used on BERT with ≤ 2 bit weights. Applying data augmentation for ≥ 8 bit BERT model is not necessary since the performance doesn't drop w.r.t to the full-precision BERT as shown in Table 5, meaning that it is able to find the optimal point without data augmentation.

4 Experiments

We measure the performance of ETSPQ on GLUE (Wang et al., 2018) and compare ETSPQ with the latest ternarization/binarization methods.

Using the V100 GPU, we fine tune with a batch size of 16 for CoLA, and 32 for other datasets and we use AdamW (Loshchilov and Hutter, 2019) with weight decay of 0.01 and learning rate of 5e-5 with a linear learning rate scheduler for five epochs.

4.1 Results

The GLUE benchmark consists of different natural language tasks. We evaluate the performance on Table 1 on the dev set, using Matthews correlation for CoLA (Warstadt et al., 2019); accuracy for SST2 (Socher et al., 2013), QNLI (Rajpurkar et al., 2016), MNLI (Williams et al., 2018), and RTE (Bentivogli et al., 2009); accuracy for MRPC (Dolan and Brockett, 2005) and QQP (Chen et al.); and, Spearman correlation for STS-B (Cer et al., 2017). Also, the average performance for all dev datasets is reported in the last column of Table 1.

Table 1 shows using our method we can ternarize BERT, compressing it by 14.9x and get a performance greater than or equal to that of the full-precision model on all GLUE development datasets, except for CoLA while using data augmentation and 8-bit activations. ETSPQ method still achieves a better performance on average compared to TernaryBERT even without data augmentation.

With binary weights and 8-bit activations, we outperform BinaryBERT (Bai et al., 2021) on all GLUE development sets. With binary weights and 4-bit activations, our ETSPQ on average outperforms BinaryBERT. At the same time, we get a reduction of 18% w.r.t (Bai et al., 2021) model size because (Bai et al., 2021) has an embedding layer twice the size as BERT embedding layer.

Ternarizing the weights and quantizing the activations to 8 bits allows us to reduce the inference flops by at least 7x while binarizing the weights and quantizing the activations to 4 bits allows us to reduce the flops by 15x. Therefore, 7x and 15x speedup will be gained for inference, respectively on CPU.

4.2 Ablation Studies

In this section, we test the performance of data augmentation under certain bitwidth settings and test the performance of different bit reduction settings. For all different cases, knowledge distillation method from (Zhang et al., 2020) is applied during fine-tuning.

4.2.1 Importance of bit reduction settings

In this section, we test out the performance of certain bit reduction settings, unlike (Zhuang et al., 2018) without using data augmentation. We test these settings on a BERT model with ternary weights and full precision activations with the same hyperparameters as before. For example, "32→2" means fine-tuning the ternary model with the initialization of the full-precision finetuned BERT.

Setting	CoLA	SST2	MRPC	STS-B	QNLI
32→16→8→4→2	55.4	93.2	88.0	89.3	91.9
32→16→8→2	53.4	93.2	87.0	89.1	91.5
32→16→2	53.6	93.1	87.0	89.2	91.4
32→8→4→2	53.7	93.1	88.0	89.3	91.9
32→8→2	55.2	93.2	87.7	89.3	91.8
32→4→2	54.7	93.2	87.7	89.3	91.8
32→2	52.7	92.8	87.0	88.6	91.4

Table 2: Results of different progressive quantization settings

Setting	CoLA	SST2	MRPC	STS-B	QNLI
32→16→8→4→2	50	205	21	36	510
32→16→8→2	40	168	16	29	403
32→16→2	30	125	12	20	303
32→8→4→2	39	170	15	28	405
32→8→2	29	123	12	21	301
32→4→2	30	124	12	20	302
32→2	20	80	8	14	203

Table 3: Runtime for different progressive quantization settings in minutes.

From Table 2, we can see that fine-tuning for more bitwidths doesn't necessarily get us more performance. For example, "32→8→4→2" has almost the same performance as "32→16→8→4→2" with "32→16→8→4→2" outperforming only on CoLA and on SST-2. "32→8→2" also has almost the same performance as "32→16→8→4→2", but "32→8→2" only falls behind on MRPC, CoLA, QNLI by at most 0.3 points.

Therefore, the full-precision model is a good initialization point when fine-tuning an 8 bit model. When the 8 bit model reaches the optimal point, the 8 bit model will also be a good intialization point when fine-tuning a 2 bit model and the resulting 2 bit model will also have a good performance. Therefore, to get the best tradeoff in terms of speed and performance, it is better to use the bit reduction setting, "32→8→2".

4.2.2 Progressively Quantize Weights First or Activations First?

Setting	CoLA	SST2	MRPC	STS-B	QNLI
32→8→2→2+8	55.4	93.2	87.7	89.3	91.8
32→32+8→8+8→2+8	55.3	93.2	87.8	89.0	91.5

Table 4: Performance of progressively reducing weight bitwidth first vs activation bitwidth first.

We also evaluate whether it is better to first progressively reduce the bitwidth of activations or to

first progressively reduce the bitwidth of weights while fine-tuning. As seen in Table 4, there is negligible performance drop when trying to first progressively reduce the bitwidth of activations. Results in Table 4 are recorded using a ternary BERT with 8 bit activations, using weight bit reduction settings, "32→8→2".

4.2.3 Importance of selective Data Augmentation

	CoLA		MRPC		SST2	
Weight Bits	-aug	+aug	-aug	+aug	-aug	+aug
16	62.0	61.5	88.1	87.9	93.6	93.8
8	62.3	62.9	88.0	88.0	94.3	94.4
4	59.0	58.5	88.0	88.0	93.9	93.7
2	54.1	58.5	87.7	88.2	93.1	93.9
1	47.2	52.6	84.5	87.3	92.4	93.5

Table 5: Results of data augmentation on different weight bitwidths.

In this section, we evaluate the importance of applying data augmentation under certain bitwidths. This helps us to determine which bitwidth to train with data augmentation to get the best performance which will then be used as initialization point when fine tuning the lower bitwidth model. From Table 5, using data augmentation for an ≥ 8 bit BERT does not improve the performance by a lot as we can see that 8 and 16 bit BERT has almost the same performance. Therefore, data augementation is best when fine-tuning on a ≤ 2 bit BERT.

5 Conclusion

In this work, we introduced an efficient two-stage progressive quantization method to solve the problem of irregular and complex landscape of BERT incurred by the binarization/ternarization of its weights and to reduce the training burden of progressive quantization. We are the first to outperform the state-of-the-art binarization method of BERT model without adding an extra module. Moreover, our ternary BERT can outperform the performance of its full-precision counterpart. Also, we can outperform the state-of-the-art BERT model with ternary weights on almost all GLUE datasets without the use of data augmentation.

References

Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jin Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2021. BinaryBERT: Pushing the limit of BERT quantization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4334–4348, Online. Association for Computational Linguistics.

Luisa Bentivogli, Bernardo Magnini, Ido Dagan, Hoa Trang Dang, and Danilo Giampiccolo. 2009. The fifth PASCAL recognizing textual entailment challenge. In *TAC*.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada. Association for Computational Linguistics.

Zihan Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. Quora Question Pairs.

ARM Cortex-M. 2020. https://developer.arm.com/ip-products/processors/cortex-m.

Mattieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training Deep Neural Networks with binary weights during propagations.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. In *Advances in Neural Information Processing Systems*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Fengfu Li, Bo Zhang, and Bin Liu. 2016. Ternary Weight Networks.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *ICLR*.

Haotong Qin, Yifu Ding, Mingyuan Zhang, Qinghua Yan, Aishan Liu, Qingqing Dang, Ziwei Liu, and

Xianglong Liu. 2022. Bibert: Accurate fully binarized bert. In *International Conference on Learning Representations (ICLR)*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, Online. Association for Computational Linguistics.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8Bit BERT. *CoRR*, abs/1910.06188.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. TernaryBERT: Distillation-aware ultra-low bit BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 509–521, Online. Association for Computational Linguistics.

Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. 2018. Towards Effective Low-Bitwidth Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

A Bert Quantization:

BERT contains N identical transformer encoder layers with the same architecture but with different parameters. A transformer layer usually contains a Multi-Head Attention (MHA) module and a Feed-Forward Network (FFN) module. Before the input goes through the transformer layers, the input first gets processed at the embedding layer as

$$\mathbf{H}_1 = \text{EM}_{\mathbf{W}^E}(\mathbf{z}) + \text{EM}_{\mathbf{W}^S}(\mathbf{z}) + \text{EM}_{\mathbf{W}^P}(\mathbf{x}) \quad (5)$$

where EM is the embedding function that uses the indices $\mathbf{z}$ to extract the word, segmentation and position embeddings from the $\mathbf{W}^E$, $\mathbf{W}^S$, $\mathbf{W}^P$ lookup tables, respectively. $\mathbf{W}^E$, $\mathbf{W}^S$, $\mathbf{W}^P$ are learnable parameters. Then the output of the embedding layer $\mathbf{H}_1$ becomes the input for the first transformer layer.

In the l-th Transformer layer, the input $\mathbf{H}_l \in \mathbb{R}^{n\times d}$ where n and d are the sequence length and hidden state size, respectively, first goes through the MHA module. In the MHA module with N_H attention heads, the head contains the parameters, $\mathbf{W}_h^Q$, $\mathbf{W}_h^K$, $\mathbf{W}_h^V \in \mathbb{R}^{d\times d_h}$ where $d_h = \frac{d}{N_H}$. Using the dot product of queries and keys from the input, the attention scores are calculated as:

$$\mathbf{A}_h = \mathbf{Q}\mathbf{K}^\top = \mathbf{H}_l\mathbf{W}_h^Q(\mathbf{H}_l\mathbf{W}_h^K)^\top \quad (6)$$

The softmax function is then applied on the attention scores to get the output of each head, head_h as

$$\text{head}_h = \text{Softmax}(\frac{\mathbf{A}_h}{\sqrt{d}})\mathbf{H}_l\mathbf{W}_h^V \quad (7)$$

The output of the multi-head attention is calculated as:

$$\text{MHA}(\mathbf{H}_l) = \text{Concat}(\text{head}_1, ..., \text{head}_{N_H})\mathbf{W}^O \quad (8)$$

A layer normalization is then applied on the output of MHA plus the input of the MHA as shown below

$$\mathbf{X}_l = \text{LN}(\text{MHA}(\mathbf{H}_l) + \mathbf{H}_1) \quad (9)$$

Then the output of that layer-normalization, $\mathbf{X}_l$, is then inputed into the FFN layer which has two linear layers containing the parameters, $\mathbf{W}^{\text{Int}} \in \mathbb{R}^{d\times 4d}$ and $\mathbf{W}^{\text{Out}} \in \mathbb{R}^{4d\times d}$. The output of FFN can be calculated as

$$\text{FFN}(\mathbf{X}_l) = \text{GeLU}(\mathbf{X}_l\mathbf{W}^{\text{Int}} + \mathbf{b}^{\text{Int}})\mathbf{W}^{\text{Out}} + \mathbf{b}^{\text{Out}} \quad (10)$$

Then, a layer normalization is then applied on the output of FFN plus the input of the FFN as shown below

$$\mathbf{H}_{l+1} = \text{LN}(\text{FFN}(\mathbf{X}_l) + \mathbf{X}_1) \quad (11)$$

Following (Zafrir et al., 2019; Zhang et al., 2020; Bai et al., 2021), we quantize the weights $\mathbf{W}^E$ from eq. 5, $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$ from eq. 6 and 7, $\mathbf{W}^O$ from eq. 8, $\mathbf{W}^{\text{Int}}$ and $\mathbf{W}^{\text{Out}}$ from eq. 10 and also quantize the inputs which will be multiplied by these weights. $\mathbf{W}^S$, $\mathbf{W}^P$ and the biases are not quantized as told in (Zhang et al., 2020; Bai et al., 2021), because these parameters' sizes are negligible and operations in the softmax, layer normalization and the task specific layer are kept in full precision because the parameters in these operations are negligible and quantizing them can significantly hurt the performance.

(Zhang et al., 2020) proposed TernaryBERT to ternarize the weights and quantized the activations to 8 bits using layer wise knowledge distilation from (Jiao et al., 2020) to transfer the knowledge of a full precision model to a ternary model by minimizing the mean-squared error (MSE) between the teacher embedding output $\mathbf{H}_1^T$ and the student embedding output $\mathbf{H}_1^S$ as shown in Eq. 12; between the teacher multi-head attention (MHA) scores $\mathbf{A}^T$ and the student MHA output $\mathbf{A}^S$ as shown in Eq. 14; and between teacher feed-forward network (FFN) $\mathbf{H}_l^T$ and the student FFN output $\mathbf{H}_l^S$ as shown in Eq. 13. And by minimizing the soft cross-entropy loss between the student's logit $\mathbf{y}^S$ and the teacher's logit $\mathbf{y}^T$ as shown in Eq. 15.

$$\mathcal{L}_{emb} = \text{MSE}(\mathbf{H}_1^S, \mathbf{H}_1^T) \quad (12)$$

$$\mathcal{L}_{trm} = \sum_{l=1}^{L} \text{MSE}(\mathbf{H}_l^S, \mathbf{H}_l^T) \quad (13)$$

$$\mathcal{L}_{att} = \sum_{l=1}^{L} \text{MSE}(\mathbf{A}_l^S, \mathbf{A}_l^T) \quad (14)$$

$$\mathcal{L}_{pred} = \text{SCE}(\mathbf{y}^S, \mathbf{y}^T) \quad (15)$$

$$\mathcal{L} = \mathcal{L}_{pred} + \mathcal{L}_{emb} + \mathcal{L}_{trm} + \mathcal{L}_{att} \quad (16)$$

Model	W-A	Size (MB)	CoLA	SST2	MRPC	STS-B	QNLI	QQP	MNLI-m	RTE	avg
Full precision	32-32	417.6	51.1	92.8	88.1/83.3	87.0/85.8	90.8	71.2/89.2	84.5	70.2	81.6
TernaryBERT	2-8	28.0	47.8	92.9	87.5/82.6	84.3/82.7	90.0	70.4/88.4	83.0	68.4	80.1
ETSPQ	2-8	28.0	**50.8**	**93.1**	**88.1/83.7**	**86.3/85.2**	**90.9**	**71.4/89.3**	**84.3**	**70.0**	**81.5**
BinaryBERT	1-8	16.5	**51.6**	91.9	85.9	82.3	89.8	89.0	**84.1**	67.3	80.2
ETSPQ	1-8	13.4	50.0	**93.0**	**87.6**	**84.8**	**90.1**	**89.1**	83.9	**67.5**	**80.8**
BinaryBERT	1-4	16.5	**47.9**	**93.1**	86.6	82.9	**89.7**	**89.0**	**83.6**	65.8	79.8
ETSPQ	1-4	13.4	46.9	93.0	**87.0**	**84.5**	89.3	**89.0**	83.5	**68.0**	**80.2**

Table 6: Quantization performance for GLUE test dataset. For binary weights, we only compare accuracy for QQP, F1 for MRPC, and Spearman Correlation for STS-B because in (Bai et al., 2021), results are only represented in these metrics.

Model	W-A	SQuAD2.0
Full precision	32-32	75.2/77.9
TernaryBERT	2-8	73.3/76.6
ETSPQ	2-8	**74.5/77.5**
BinaryBERT	1-8	73.6/76.5
ETSPQ	1-8	**73.9/76.8**
BinaryBERT	1-4	**72.5/75.4**
ETSPQ	1-4	72.1/75.2

Table 7: Our performance vs state of the art methods for SQuAD datasets.

Task	-aug	+aug
CoLA	8.5k	220k
SST2	67k	1135k
MRPC	3.7k	226k
STS-B	5.7k	327k
QNLI	108k	4278k
RTE	2.5k	147k

Table 8: Size of the dataset for each task with and without data augmentation from (Jiao et al., 2020)

Task	-aug	+aug
CoLA	2.2	52
SST2	8.0	129
MRPC	1.1	52
STS-B	1.4	75
QNLI	20	403
RTE	0.6	30

Table 9: Runtime in minutes for one epoch for each task with and without data augmentation from (Jiao et al., 2020)

B GLUE Test Results

On GLUE test set, we can also achieve a comparable performance w.r.t. the full-precision baseline, only losing around 0.1 % of the full precision average performance with ternary weights and 8-bit activations according to Table 6. With binary weights and 8-bit activations, we outperform BinaryBERT (Bai et al., 2021) on all GLUE test set except for MNLI and CoLA.

C SQuAD 2.0 Results

We also perform experiments to measure the performance of ETSPQ on SQuAD 2.0 (Rajpurkar et al., 2016) datasets and compare with state-of-the-art quantized models. From Table 7, using ternary weights we outperform the performance of (Zhang et al., 2020). With binary weights, we can outperform the performance of (Bai et al., 2021) for 8 bit activations, but for 4 bit activations we achieved a comparable performance.

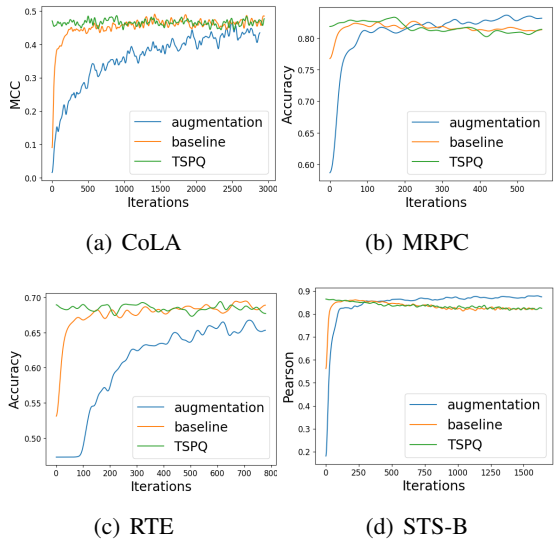
D Further Analysis

In this section, we study the real impact of data augmentation and Two-Stage Progressive Quantization (TSPQ) on the performance of quantized BERT with binary weights and 4-bit activations by plotting the performance on the dev set over the number of iterations for the case "Augmentation" where we use only data augmentation and use the full precision fine-tuned model initialization; for the case "TSPQ" where we use the slightly higher precision quantized fine-tuned model initialization; and for the case "Baseline" where we only use the full precision fine-tuned model initialization.

Fine-tuning with data augmentation for one epoch takes longer and takes more iterations than fine-tuning without data augmentation for five epochs on a GLUE task due to the larger size of the augmented data.

Therefore, the model running with more epochs has more opportunities to adjust its weights properly under the quantization constraint and it is unfair to just compare the effects of using data augmentation for one epoch vs five epochs without data augmentation as previous works did.

As a result of that, we made sure that the quantized model is being fine-tuned for the same num-



(a) CoLA (b) MRPC

(c) RTE (d) STS-B

Figure 2: Performance over training iterations on CoLA (a), MRPC (b), RTE (c), and STS-B (d) on dev set for binary weights and 4 bit activations.

ber of iterations and not necessarily the same number of epochs for all three cases as illustrated in Figure 2. Knowledge distillation from (Zhang et al., 2020) and activation quantization function from Eq. 3 is used in all 3 cases

Figure 2 shows that the use of data augmentation is not beneficial across all datasets unless fine-tuning the model for a large number of iterations. For instance, the performance increase for data augmentation is a lot more progressive for RTE and CoLA datasets. In this case, fine-tuning can be performed for the non-augmented datasets with significantly fewer iterations to reach optimal performance. On the other hand, the use of data augmentation helps to improve the performance of MRPC and STS-B datasets when fine-tuning the model for a larger number of iterations.

Using ETSPQ, we get a good initial performance for the target bit precision, whereas training it normally would need a few more iterations to approximately match the performance. Figure 2 shows that the trajectory of fine-tuning the quantized BERT using ETSPQ is similar to the Baseline at the later iterations and the curve from ETSPQ is generally above or at a equal height as the curve of the Baseline.

E Comparing with BiBERT

Due to limited space, we do not include BiBERT's results into the main results section. Also BiBERT's method deals with the problems of binarizing activations and weights by replacing the softmax function with a boolean function and by replacing the mean square error (MSE) distillation with the direction mismatch distillation (DMD) for the distillation of the attention scores.

Task	DMD	MSE
CoLA	52.1	52.3
SST2	92.8	92.5
MRPC	86.5	87.0
STS-B	88.8	88.6
QNLI	91.4	91.4

Table 10: Performance of using Direction Mismatch Distillation (DMD) versus using Mean Square Error (MSE) Distillation.

However, using a boolean function does not work when the output of the softmax is supposed to be ≥ 1 bit. Therefore, this is also one of the reasons it wasn't included in the main results and we only compare the performance of using DMD with the performance of using MSE distillation in Table 10.

In Table 10, we can see that the performance doesn't change a lot when using DMD compared to using MSE distillation. Results obtained in the table is obtained using the full-precision BERT as the starting point and finetuning a BERT model with ternary weights and 8 bit activations.