Q8BERT: Quantized 8Bit BERT

Ofir Zafrir **Guy Boudoukh** **Peter Izsak** **Moshe Wasserblat**
Intel AI Lab
{ofir.zafrir, guy.boudoukh, peter.izsak, moshe.wasserblat}@intel.com

Abstract

Recently, pre-trained Transformer [14] based language models such as BERT [3] and GPT [9], have shown great improvement in many Natural Language Processing (NLP) tasks. However, these models contain a large amount of parameters. The emergence of even larger and more accurate models such as GPT2 [8] and Megatron[1], suggest a trend of large pre-trained Transformer models. However, using these large models in production environments is a complex task requiring a large amount of compute, memory and power resources. In this work we show how to perform quantization-aware training during the fine-tuning phase of BERT in order to compress BERT by $4\times$ with minimal accuracy loss. Furthermore, the produced quantized model can accelerate inference speed if it is optimized for 8bit Integer supporting hardware.

1 Introduction

Pre-trained transformer language models (GPT [9], XLNet [17], XLM [7], BERT [3]) have demonstrated State-of-the-Art (SOTA) results for a variety of NLP tasks such as sentence classification, sequence tagging and question answering, by extracting contextual word representations or by fine-tuning the whole model on a target task. The models are pre-trained on extremely large corpora and result in a large number of parameters. For example, Devlin et al. [3] introduced two pre-trained models: BERT-Base, which has 110M parameters in 32bit Floating Point (FP32) representation, and BERT-Large, which has 334M parameters in FP32 representation. Both BERT models have a high memory footprint and require heavy compute and wide bandwidth during inference. In addition, real time NLP applications that integrate BERT have to meet low latency requirements to achieve a high quality customer experience, therefore, the computational characteristics of BERT pose a challenge to deployment in production environments. These models will have a major impact on the way business organizations consume computing resources, since computing resources will have to handle loading of large models and heavy feed-forward calculations, shifting workload focus from lower level training to more application-specific fine-tuning and inference. Therefore, it is crucial to develop energy-efficient and minimum-cost methods to run these models in production [16].

Model compression algorithms are used to reduce compute and memory resources required for running inference. For example, Han et al. [4] used a pipeline of pruning, quantization and Huffman encoding in order to achieve a compression ratio of $49\times$ of VGG-16 [11]. As a result, the compressed VGG-16 can be fitted into an on-chip SRAM cache which allows faster access times with less power in comparison to off-chip DRAM memory. In another example, Jacob et al. [5] introduced a method of training linear quantized Convolutional Neural Networks (CNN) that uses Integer Arithmetic instead of Floating Point Arithmetic which can be up to $4\times$ faster using only 25% of the memory footprint [13].

In this work, we present a method for achieving best-in-class compression-accuracy ratio for BERT. To do this, we apply quantization-aware training during the fine-tuning process of BERT. We quan-

[1] https://github.com/NVIDIA/Megatron-LM

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

tize all GEMM (General Matrix Multiply) operations in BERT Fully Connected (FC) and Embedding layers. We simulate 8bit quantized inference while maintaining 99% accuracy in comparison to the FP32 version of BERT for eight different NLP tasks. Moreover, since we quantize all the FC and Embedding layers' weights - which comprise over 99% of the model's weights - to 8bit, we achieve a memory footprint $4\times$ smaller than the original BERT. In addition, it is possible to use our method to implement efficient inference with hardware that supports 8bit arithmetic and an optimized library for 8bit GEMM. We have released our work as part of our open source model library `NLP Architect`[2].

The method presented in this paper is not exclusive to BERT model and can be integrated into other large pre-trained Transformer based models.

2 Method

In this section, we describe the quantization scheme, linear quantization, and quantization-aware training method we used. We chose to use this quantization scheme because, in addition to reducing the model size by approximately $4\times$, it is also possible to accelerate inference time by using Integer arithmetic to calculate GEMM using specialized hardware for Integer and Fixed Point calculations. For example, Bhandare et al. [2] stated that using Intel® Xeon® Cascade Lake's Vectorized Neural Network Instructions (VNNI) to perform Int8 matrix multiplication provides a speed-up of $3.7\times$ over FP32 matrix multiplication. Moreover, by using symmetric linear quantization we simplify the quantization process and zero out terms related to the offset part of the quantized values. Our method is based on the method proposed by Jacob et al. [5].

2.1 Quantization Scheme

We use symmetric linear quantization as our quantization scheme for quantizing both weights and activations to 8bit Integers (Int8):

$$\begin{aligned} \text{Quantize}(x|S^x, M) &:= \text{Clamp}\left(\lfloor x \times S^x \rceil, -M, M\right), \\ \text{Clamp}\left(x, a, b\right) &= \min\left(\max\left(x, a\right), b\right) \end{aligned} \tag{1}$$

where S^x is the quantization scaling-factor for input x and M is the highest quantized value when quantizing to b number of bits:

$$M = 2^{b-1} - 1 \tag{2}$$

E.g. when quantizing to 8 bits, $M = 127$. The scaling-factor can be determined either dynamically during inference, or calculated using statistics collected during training, or calculated using statistics collected, post-training, during inference on a calibration set. In our work the weights' scaling-factor is calculated according to:

$$S^W = \frac{M}{\max\left(|W|\right)} \tag{3}$$

and the activations' scaling-factor is calculated based on values seen during training using an Exponential Moving Average (EMA):

$$S^x = \frac{M}{\text{EMA}\left(\max\left(|x|\right)\right)} \tag{4}$$

2.2 Quantized-Aware Training

Quantization-aware training is a method of training Neural Networks (NN) to be quantized at the inference stage, as opposed to post-training quantization where the training is executed without any adaptation to the quantization process. In our work, we use fake quantization to introduce the quantization error to the model during the training phase in order for the model to learn to bridge the quantization error gap. Fake quantization is an operation that simulates the rounding effect in Floating Point values as presented by Jacob et al. [5]. Since the rounding operation is not derivable, we use the Straight-Through Estimator (STE) [1] to estimate the gradient of fake quantization:

$$\frac{\partial x^q}{\partial x} = \vec{\mathbf{1}} \tag{5}$$

[2] `https://github.com/NervanaSystems/nlp-architect`

Table 1: GLUE tasks and SQuAD results. Each score is evaluated on the publicly available development set for the task using the metric specified for each task. For each task we present the score of a baseline (FP32) model, of a Quantization-Aware Training (QAT) model quantized to 8bit, and of a Dynamically Quantized (DQ) to 8bit model. Large means those tasks were trained with BERT-Large architecture.

Dataset	Metric	BERT baseline accuracy (STD)	QAT BERT 8bit (STD)	DQ BERT 8bit (STD)
CoLA	Matthew's corr.	**58.48** (1.54)	**58.48** (1.32)	56.74 (0.61)
MRPC	F1	**90** (0.23)	89.56 (0.18)	87.88 (2.03)
MRPC-Large	F1	90.86 (0.55)	**90.9** (0.29)	88.18 (2.19)
QNLI	Accuracy	90.3 (0.44)	**90.62** (0.29)	89.34 (0.61)
QNLI-Large	Accuracy	91.66 (0.15)	**91.74** (0.36)	88.38 (2.22)
QQP	F1	87.84 (0.19)	**87.96** (0.35)	84.98 (0.97)
RTE	Accuracy	**69.7** (1.5)	68.78 (3.52)	63.32 (4.58)
SST-2	Accuracy	**92.36** (0.59)	92.24 (0.27)	91.04 (0.43)
STS-B	Pearson corr.	**89.62** (0.31)	89.04 (0.17)	87.66 (0.41)
STS-B-Large	Pearson corr.	**90.34** (0.21)	90.12 (0.13)	83.04 (5.71)
SQuADv1.1	F1	**88.46** (0.15)	87.74 (0.15)	80.02 (2.38)

where x^q denotes the result of fake quantizing x. Using the combination of fake quantization and STE we are able to perform quantized inference during training while back-propagating at full precision which allows the FP32 weights to overcome the quantization error.

3 Implementation

Our goal is to quantize all the Embedding and FC layers in BERT to Int8 using the method described in Section 2. For this purpose we implemented quantized versions of Embedding and FC layers. During training, the Embedding layer returns fake quantized embedding vectors, and the quantized FC performs GEMM between the fake quantized input and the fake quantized weight, and then accumulates the products to the bias which is untouched since the bias will be later quantized to Int32. During inference, the quantized Embedding layer returns Int8 embedding vectors, and the quantized FC performs GEMM between Int8 inputs accumulated to the Int32 bias which is quantized using the weights' and activations' scaling-factors as described in [5]. Although the bias vectors are quantized to Int32 values, they only make up for a fraction of the amount of parameters in the model.

Our implementation of Quantized BERT is based on the BERT implementation provided by the `PyTorch-Transformers`[3] library. To implement quantized BERT we replaced all the Embedding and FC layers in BERT to the quantized Embedding and FC layers we had implemented. Operations that require higher precision, such as Softmax, Layer Normalization and GELU, are kept in FP32.

4 Evaluation

To test our approach we evaluated our model on the GLUE (General Language Understanding Evaluation) benchmark [15], which is a collection of resources for training, evaluating, and analyzing natural language understanding systems in a wide array of NLP tasks. The ultimate goal of GLUE is to drive research in the development of general and robust natural language understanding systems. In addition, we evaluated our model on the question and answering task SQuADv1.1 [10]. The Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowd workers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage.

We summarized our results for quantized BERT in the QAT column in Table 1. We ran each experiment five times and reported the average result and standard deviation. In addition, we calculated the relative error induced by the quantization process and summarized the results in Table 2. In all experiments we used BERT-Base as the base model unless indicated otherwise. In all experiments

[3] `https://github.com/huggingface/pytorch-transformers`

Table 2: Reduction in accuracy induced by quantization relative to baseline model. DQ is the Dynamically Quantized model and QAT is the Quantization-aware Trained quantized model. Large means those tasks were trained with BERT-Large architecture.

Method	CoLA	MRPC	MRPC-Large	QNLI	QNLI-Large	QQP	RTE	SST-2	STS-B	STS-B-Large	SQuADv1.1
DQ	2.98%	2.36%	2.95%	1.69%	3.58%	1.85%	9.13%	1.43%	2.19%	8.08%	9.54%
QAT	0.00%	0.49%	-0.04%	-0.35%	-0.09%	-0.14%	1.32%	0.13%	0.65%	0.24%	0.81%

we fine-tuned the pre-trained models offered by Tensorflow-Hub[4]. In our internal testing, we found that the relative error induced by quantization is less than 1% (excluding RTE task) while the space capacity of the model is reduced by 4x.

4.1 Effect of Quantization-Aware Training

In order to measure the necessity of quantization-aware training we compared our results to post-training quantized models. We quantized our baseline models using Dynamic Quantization (DQ). The weights and activations are quantized as described in Section 2.1 with a small difference in the way we calculate the quantization scaling-factor of the activations. Instead of using Equation 4 we compute the scale the same way we compute the weights' scaling-factor using Equation 3. This calculation is done during inference for each incoming activation tensor. The results for this quantization method are summarized in the DQ column in Table 1 and the relative error induced by quantization is also summarized in Table 2. We observe that the DQ method produces significantly worse results over all tasks.

5 Related Work

Compressing Transformer-based models for efficient inference is an active field of research. Junczys-Dowmunt et al. [6] applied knowledge distillation and 8bit post-training quantization to speed up Transformer models for neural machine translation (Transformer-LT) [14], however, the quantized model suffered a loss of 1 BLEU score in comparison to the baseline model. Bhandare et al. [2] also applied 8bit post-training quantization to Transformer-LT models and demonstrated how to utilize Intel® specialized 8bit hardware to accelerate the inference process. Habana Labs[5] published Quantized BERT performance measurements on its in-house accelerator for NN inference, however, Habana quantized BERT to 16bit Integer which offers a much wider quantization range and only $2\times$ compression. NVIDIA[6] also measured BERT performance on its in-house accelerator using 16bit Floating Point arithmetic. Furthermore, NVIDIA implemented a number of optimized kernels for BERT's operations in order to save memory bandwidth during inference. Sucik [12] fine-tuned BERT on a custom dataset and performed 8bit Integer post-training quantization.

6 Conclusions and Future Work

We have shown a method for quantizing BERT GEMM operations to 8bit for a variety of NLP tasks with minimum loss in accuracy, and hope that the software developers community can use our quantization method to compress BERT and implement efficient BERT inference with 8bit GEMM operations. Efficient inference will enable low-latency NLP applications on a variety of hardware platforms from edge devices to data centers. In the future we intend to apply other model compression methods in order to compress BERT. Decreasing BERT's memory footprint will accelerate BERT inference time and reduce power consumption, both of which are critical for deploying BERT in production environments having low memory and power resources. Furthermore, we intend to integrate other compression methods with our quantized BERT model.

References

[1] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] `https://www.tensorflow.org/hub`

[5] `https://habana.ai/habana-labs-goya-delivers-inferencing-on-bert/`

[6] `https://devblogs.nvidia.com/nlu-with-tensorrt-bert/`

[2] A. Bhandare, V. Sripathi, D. Karkada, V. Menon, S. Choi, K. Datta, and V. Saletore. Efficient 8-bit quantization of transformer neural machine language translation model. *arXiv preprint arXiv:1906.00532*, 2019.

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2018.

[4] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[5] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[6] M. Junczys-Dowmunt, K. Heafield, H. Hoang, R. Grundkiewicz, and A. Aue. Marian: Cost-effective high-quality neural machine translation in c+. *ACL 2018*, page 129, 2018.

[7] G. Lample and A. Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019.

[8] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners.

[9] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[10] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. 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, 2016.

[11] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[12] S. Sucik. Compressing bert for faster prediction. *URL https://blog.rasa.com/compressing-bert-for-faster-prediction-2/*, 2019.

[13] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

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

[15] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. 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, 2018.

[16] M. Wasserblat, O. Pereg, and G. Singer. Future directions for nlp in commercial environments. *URL https://www.intel.ai/future-directions-nlp*, 2019.

[17] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.