Towards Efficient Post-training Quantization of Pre-trained Language Models

Haoli Bai\(^1,2\), Lu Hou\(^1\), Lifeng Shang\(^1\), Xin Jiang\(^1\), Irwin King\(^2\), Michael R. Lyu\(^2\)
\(^1\)Huawei Noah’s Ark Lab, \(^2\)The Chinese University of Hong Kong
\{baihaoli,houlu3,Shang.Lifeng,Jiang.Xin\}@huawei.com,
\{king,lyu\}@cse.cuhk.edu.hk

Abstract

Network quantization has gained increasing attention with the rapid growth of large pre-trained language models (PLMs). However, most existing quantization methods for PLMs follow quantization-aware training (QAT) that requires end-to-end training with full access to the entire dataset. Therefore, they suffer from slow training, large memory overhead, and data accessibility issues. In this paper, we study post-training quantization (PTQ) of PLMs, and propose module-wise quantization error minimization (MREM), an efficient solution to mitigate these issues. By partitioning the PLM into multiple modules, we minimize the reconstruction error incurred by quantization for each module. In addition, we design a new model parallel training strategy such that each module can be trained locally on separate computing devices without waiting for preceding modules, which brings nearly the theoretical training speed-up (e.g., \(4\times\) on 4 GPUs). Experiments on GLUE and SQuAD benchmarks show that our proposed PTQ solution not only performs close to QAT, but also enjoys significant reductions in training time, memory overhead, and data consumption.

1 Introduction

Large pre-trained language models (PLMs) have achieved remarkable success in various natural language processing tasks \([44, 12, 9]\). However, the increasing size and computation overhead also make it prohibitive to deploy these PLMs on resource-constrained devices. To obtain compact PLMs, various model compression methods have been proposed, such as pruning \([32, 14, 50]\), knowledge distillation \([40, 42, 24]\), weight-sharing \([10, 25, 46, 22]\), dynamic computation with adaptive depth or width \([18, 52, 59]\), and quantization \([55, 41, 54, 56, 6, 48, 43]\).

Among these methods, network quantization enjoys the reduction of both model size and computation overhead without modifying the network architecture, and is thus extensively studied in PLM quantization \([55, 41, 54, 56, 6, 48, 43]\). However, most previous methods for PLM quantization methods follow quantization-aware training (QAT), which suffers from several issues. Specifically, QAT usually conducts end-to-end back-propagation training over the whole training set, which can be slow in training time, memory demanding and data consuming. These issues can be sometimes prohibited for industrial PLMs. Post-training quantization (PTQ), on the other hand, serves as an appealing alternative that is efficient in training time, memory overhead and data consumption. Instead of the full training set, PTQ can leverage only a small portion of training data to minimize the layer-wise reconstruction error incurred by quantization \([43, 55, 53, 23]\). This can be done by either calibrating the batch normalization statistics \([42]\) or step sizes \([35]\) in quantization functions. The layer-wise objective also breaks down the end-to-end training, making the prob-

\(^\star\)This work is partially done during internship at Huawei Noah’s Ark Lab.

lem more sample-efficient and memory-saving. Nonetheless, we find it non-trivial to directly apply previous PTQ methods for PLMs such as BERT, as the performance drops sharply.

In this paper, we aim at improving the performance of post-training quantization for PLM, while simultaneously maintaining its efficiency w.r.t training time, memory overhead and data accessibility. First, we partition the PLM into multiple modules, and propose module-wise reconstruction error minimization (MREM). Each module consists of multiple Transformer layers, which permits the sufficient optimization than the previous layer-wise objective. Meanwhile, the module size can be flexibly adjusted for the proper trade-off between layer-wise correlation and memory overhead of computing devices. While a similar block-wise objective is previously considered in [29], they require second-order Hessian matrices for optimization, which can be computationally prohibitive for large PLMs. Second, we design a new model parallel strategy to further accelerate the training process of MREM. By allocating each partitioned module to the individual computing device, our design allows all modules to be trained jointly without synchronizing with adjacent partitions. This brings nearly the theoretical speed-up (e.g., 4× on 4 GPUs), superior to previous data parallel [29] or model parallel [21,36] techniques. Third, we develop annealed teacher forcing for our parallel strategy. We find that the naive parallel training suffers from the propagation of reconstruction error, since each quantized module passes the error to its successors before it is fully optimized. Inspired by [51], we use the full-precision module to provide clean signals to the next quantized module. This breaks the reconstruction error propagation and further improves the performance of parallel strategy.

We summarize the contributions of this paper as follows (i) We study the post-training quantization of PLMs, and propose module-wise reconstruction error minimization (MREM), a fast, memory-saving, and data-efficient approach to improve the quantized PLMs. (ii) We design a new model parallel strategy based on MREM to accelerate post-training quantization with theoretical speed-up for distributed training. (iii) The parallel MREM can be combined with annealed teacher forcing to alleviate the propagation of reconstruction error and boost the performance. (iv) Empirical results on the GLUE and SQuAD benchmarks demonstrate superiority of our PTQ solution over QAT. For instance, Figure 1 shows that the BERT-large model trained by parallel MREM achieves 85.5% accuracy with only 4K training samples. Moreover, it consumes merely one-third of memory per GPU and is more than 150× faster than previous QAT training.

2 Motivation

2.1 Quantization Background

Network quantization replaces the original full-precision weight or activation $x \in \mathbb{R}^{m \times n}$ with its lower-bit counterpart $\hat{x}$. Denoting $s \in \mathbb{R}^+$ as the step size, the $b$-bit symmetric uniform quantization function $Q_b(\cdot)$ can be written as

$$\hat{x} = Q_b(x) = s \cdot \Pi(\Omega(b))(x/s),$$

where $\Omega(b) = \{-2^{b-1} + 1, ..., 0, ..., 2^{b-1} - 1\}$ is the set of $b$-bit integers, and $\Pi(\cdot)$ is the projection function that maps $x/s$ to its closest integer.

To quantize Transformer-based PLMs, we follow the setting in previous works [5,56]: we quantize both the network weights and activations in each matrix multiplication. We use symmetric uniform quantization for weights, embeddings, and activations, except activations after the self-attention and GeLU function. For these two activations, we adopt asymmetric quantization since their elements are mostly positive. We skip the quantization for all layer-normalization layers, skip connections, biases, etc.
Figure 2: Comparison between QAT and REM-based PTQ over four dimensions. We use a BERT-large model over MNLI dataset for illustration. The full-precision (FP) fine-tuning is also included as a baseline. We follow the procedure in [56] for QAT, and REM in Equation (2) for PTQ. The training time and memory in (a) and (b) are measured by 4-bit weights and 8-bit activations (i.e., W4A8) on an NVIDIA V100 GPU.

and the last classification head due to little computation overhead or large performance drop. We leave more details on the choice of quantization functions and their algorithmic details in Appendix A.

Post-training Quantization. Recently popular PTQ methods [35, 17, 33, 23, 30] constructs a tiny unlabeled dataset (a.k.a. calibration set) \( \tilde{D} \) from the original training data \( D \). These approaches usually aim at reconstruction error minimization (REM), i.e., minimizing the distance between the output of multiplication between the quantized and the full-precision counterpart as follows:

\[
\min_{w, a} \| \hat{w}^T \hat{a} - w^T a \|^2, \quad \text{s.t.} \quad \hat{w} = Q_{b_w}(w), \quad \hat{a} = Q_{b_a}(a),
\]

where \( w \) and \( a \) are full-precision weights and activations, \( \hat{w} \) and \( \hat{a} \) are their quantized representations with \( b_w \) and \( b_a \) bit-widths, and \( s \) denotes all step-sizes involved for quantization. REM is usually conducted in a greedy manner. It proceeds to the matrix multiplication only after the training of previous ones. The simple objective in Equation (2) is lightweight to solve over \( \tilde{D} \), and is also theoretically verified to be more sample-efficient than end-to-end training [58]. While there are also other data-free PTQ variants [57, 24, 16, 54], REM-based solutions usually demonstrate better empirical results [33, 23]. Thus we mainly focus on the REM-based PTQ for PLMs in this paper.

2.2 Quantizing Pre-trained Language Models: QAT or PTQ?

To conceptually study the difference between QAT and REM-based PTQ over popular PLMs, we consider the following four dimensions, and summarize the results in Figure 2.

**Training Time.** As QAT conducts training over the full training set \( D \), it takes much more time than PTQ over the calibration set \( \tilde{D} \). Recent QAT methods [56, 3] further combine two-stage knowledge distillation [24], which can take nearly four times longer than FP, as shown in Figure 2(a).

**Memory Overhead.** The increasing size of large PLMs makes it prohibited to conduct QAT on memory-limited devices. From Figure 2(b), QAT [56] even consumes 8.3 GB more memory than FP when armed with knowledge distillation. On the contrary, PTQ only caches intermediate results in Equation (2), which can be fed into a single GTX 1080 Ti.

**Data Accessibility.** Data accessibility can be often prohibited due to data security or privacy issues in the industry. Unlike QAT that needs the full training set \( D \), PTQ constructs the calibration data \( \tilde{D} \subseteq D \) by sampling only \( 1K \sim 4K \) instances from \( D \), as shown in Figure 2(c).

**Performance.** QAT usually demonstrates superior quantization results than PTQ, and this holds true for PLMs. From Figure 2(d), QAT remains steady across different bit-widths, while REM-based PTQ drops sharply. Thus the poor performance of PTQ is the main challenge in quantizing PLMs.

In summary, REM-based PTQ is superior to QAT with regard to training efficiency, memory overhead, and data accessibility. Nevertheless, it is still often less preferred than QAT due to its severe performance drop especially for low quantization bit-width [55, 41, 56].
We partition both the full-precision model and quantized model into multiple modules and put these modules on different computing devices. By sampling tensors from the input queue, each module can be trained locally without waiting for its predecessors. Teacher forcing is applied to mitigate the issue of reconstruction error propagation on the quantized module.

3 Methodology

In this section, we aim at improving the performance of post-training quantization for PLMs, while preserving its merits of fast training, light memory overhead, and data consumption. We extend the existing reconstruction error minimization from the layer-wise to the module-wise granularity to fit Transformer models. Based on the module partition, we then design a new parallel training strategy that further speeds up the PTQ pipeline. An overview of our solution can be found in Figure 3.

3.1 Module-wise Reconstruction Error Minimization

We propose a new PTQ solution called *module-wise reconstruction error minimization* (MREM) for PLMs. Existing REM \[^{33}\] solves Equation 2 for each matrix multiplication. However, a standard transformer layer in PLMs consists of a Multi-Head Attention (MHA) and a Feed-Forward Network (FFN), both of which contain a number of matrix multiplications that are coupled together. Greedily tackling each matrix multiplication in REM can thus lead to suboptimal quantized networks. Moreover, the insufficiently minimized reconstruction error shall propagate and enlarge along with transformer layers, and finally deteriorate the network output \[^{6, 2}\].

Towards that end, the proposed module-wise reconstruction error minimization admits larger granularity by jointly optimizing all the coupled linear layers inside each module. Specifically, given a PLM with \(L\) transformer layers, embedding layers and the classification head, we partition them into \(N\) modules, where the \(n\)-th module include \([l_n, l_{n+1})\) transformer layers with \(l_n\) being the first layer of this module. MREM aims at minimizing the joint reconstruction errors between all quantized FFN output \(\hat{f}_l\) in the module from their full-precision counterpart \(f_l\) as follows:

\[
\min_{w_n, s_n} \ell(n) = \sum_{l \in [l_n, l_{n+1})} \| \hat{f}_l - f_l \|^2, \tag{3}
\]

where \(w_n\) and \(s_n\) are all learnable parameters and quantization step sizes in the \(n\)-th module. Similar to REM, MREM can be optimized sequentially: given previously trained modules, only parameters and quantization step sizes in the current module are updated. Besides the grouped Transformer layers, we also minimize the MSE loss in the Transformer embedding and output logits respectively.

Note that the number of modules \(N\) can be adjusted depending on the memory constraint of computing resources. When \(N = 1\), this reduces to intermediate-layer knowledge distillation \[^{24}\], which can be memory-demanding when quantizing large PLMs on a single GPU.

3.2 Accelerated Parallel Training

Based on the proposed MREM, we propose a new model parallel strategy to further accelerate the training. As shown in Figure 3, we put different modules on individual computing devices. A set of

\[^{2}\] The embedding layers and the classification head are incorporated in the first and last module respectively.
input queues \( \mathcal{I} = \{ \mathcal{I}_1, \ldots, \mathcal{I}_{N-1} \} \) is deployed between each pair of adjacent modules. For the \( n \)-th module, the queue collects its output of the most recent \( t_0 \) steps, i.e., \( \mathcal{I}_n^t = \{ f_{l_n}^{t_0}, f_{l_n}^{t_0-1}, \ldots, f_{l_n}^{t_0-t_0+1} \} \). Meanwhile, the \((n+1)\)-th module can always sample with replacement \( f_{l_n} \sim \mathcal{I}_n^t \) from the queue without waiting for the \( n \)-th module. Similar rules hold for the quantized module and their input queues \( \hat{\mathcal{I}} \). The design of the input queue resembles stale synchronous parallel \[17\] that caches the stale output locally so as to reduce the waiting time among workers, where \( t_0 \) is the stale threshold.

The training workflow is as follows. Initially, every module is computed one after another in the first \( t_0 \) step to fill in the input queue, after which parallel training takes place. Then the module samples input from the queue and calculates the loss \( \ell(n) \) correspondingly for \( n = 1, \ldots, N \). Meanwhile, the input queue is updated with the rule of first-in-first-out throughout the training. In the backward pass, we constrain the gradients to propagate locally within each module, without affecting its predecessors. Such a design can avoid the load imbalance issue from straggler modules, bringing nearly the theoretical \( N \times \) speed-up.

The proposed parallel training strategy is superior to the conventional data parallel training in quantizing PTMs. Meanwhile, it is also different from pipeline parallelism \[21\]. We leave the detailed comparisons with these parallelism techniques in Appendix E.

### 3.3 Annealed Teaching Forcing

Since all modules get optimized simultaneously instead of the sequential manner, the next module takes the output from the queue before its predecessor is fully optimized. Therefore, the predecessor’s reconstruction error is propagated to the following modules before it is sufficiently minimized.

Inspired by teacher forcing \[51\] in training recurrent networks, the output \( f_{l_n} \) from the \( n \)-th full-precision module naturally serves as the clean input to the \((n+1)\)-th quantized module to substitute \( \hat{f}_{l_n} \). Thus \( f_{l_n} \) stops the propagation of the accumulated error on the quantized module. Nevertheless, such an approach breaks the connection to previous quantized modules and may suffer from forward inconsistency between training and inference \[2\] on the quantized model. For the trade-off, we take the convex combination between the full-precision \( f_{l_n} \) and quantized \( \hat{f}_{l_n} \) as follows:

\[
\hat{f}_{l_n} = \lambda f_{l_n} + (1 - \lambda) \hat{f}_{l_n}, \quad \lambda \in [0, 1],
\]

where the hyperparameter \( \lambda \) controls the strength of teacher forcing. \( \lambda = 1 \) gives the full correction of reconstruction error but with forward inconsistency, while \( \lambda = 0 \) reduces to the conventional setting that suffers from the propagated reconstruction error. We adopt a linear decay strategy for \( \lambda \):

\[
\lambda_t = \max(1 - \frac{t}{T_0}, 0),
\]

where \( T_0 \) is the preset maximum steps of the decay. Intuitively, a large \( \lambda \) is desired at the beginning when each module is rarely optimized. Later, a small \( \lambda \) is preferred to transit to normal training such that the forward inconsistency can be bridged. The remaining \( T - T_0 \) steps stick to normal training so that each quantized module adapts to its own predecessors.

Finally, an overview of the proposed parallel module-wise reconstruction error minimization with annealed teacher forcing is shown in Algorithm 1 and Algorithm 2. The Update(·) in Algorithm 2 can be any gradient update function such as AdamW \[51\] with learning rate \( \eta^t \).

**Algorithm 1** Efficient PTQ for PLMs.

1: procedure Main():
2: Partition the PLM into \( N \) modules.
3: Fill in the input queues \( \mathcal{I}, \hat{\mathcal{I}} \).
4: for \( n \leftarrow 1, \ldots, N \) do
5: \( \triangleright \) run in parallel.
6: \( \quad \) while \( t < T \) do
7: \( \quad \) \( f_{l_n-1} \sim \mathcal{I}_{n-1}^t, \ f_{l_n-1} \sim \hat{\mathcal{I}}_{n-1}^t \)
8: \( \quad \) \( f_{l_n}^t, \hat{f}_{l_n}^t \leftarrow \text{MREM}(f_{l_n-1}^t, \hat{f}_{l_n-1}^t, t) \).
9: \( \quad \) Update \( \mathcal{I}_n, \hat{\mathcal{I}}_n \) with \( f_{l_n}^t, \hat{f}_{l_n}^t \).
10: Return the Quantized PLM.

**Algorithm 2** MREM algorithm.

1: procedure MREM \((f_{l_n-1}^t, \hat{f}_{l_n-1}^t, t)\):
2: \( \quad \) if \( t < T_0 \) then
3: \( \quad \) \( \lambda_t \leftarrow \max(1 - \frac{t}{T_0}, 0) \).
4: \( \quad \) Compute \( f_{l_n-1}^t \) by Equation 4.
5: \( \quad \) Compute the full-precision module output \( f_{l_n}^t \).
6: \( \quad \) Compute the quantized module output \( \hat{f}_{l_n}^t \).
7: \( \quad \) Compute the loss \( \ell(n) \) by Equation 4.
8: \( \quad \) \( w_{n+1}^{t+1} \leftarrow \text{Update}(w_{n+1}^t, \frac{\partial \ell(n)}{\partial w_n}, \eta^t) \).
9: \( \quad \) \( s_{n+1}^{t+1} \leftarrow \text{Update}(s_{n+1}^t, \frac{\partial \ell(n)}{\partial s_n}, \eta^t) \).
10: Return \( f_{l_n}^t, \hat{f}_{l_n}^t \).
4 Experiments

In this section, we empirically verify the proposed MREM for post-training quantization of PLMs. We first introduce the experimental setup in Section 4.1. Then we present main results in Section 4.2, including comparisons with QAT and REM, and other existing quantization baselines. In Section 4.3, we provide discussions on a variety of factors in our approach, such as the effect of teacher forcing, the number of model partitions, and calibration data size. Our implementation is based on MindSpore.

4.1 Experimental Setup

Datasets and Metrics. We evaluate post-training quantization on both the GLUE and SQuAD benchmarks. The size of calibration data is by default $|\tilde{D}| = 4,096$, by randomly sampling instances from the full training set. As both RTE and MRPC tasks in the GLUE benchmark contain fewer than 4,096 samples, we use their full training set on these two tasks. We leave the study of data size in Section 4.4. Each experiment is repeated ten times with different calibration sets, and both the mean and standard deviations are reported.

We use the same evaluation metrics in [12, 56] for the development set of GLUE and SQuAD benchmarks. For results in Section 4.2, we report accuracies on both the matched section and mis-matched sections of MNLI, and EM (exact match) and F1 score for SQuAD. Additionally, we also report the training time (min), memory overhead (GB) as well as the size of the training set (K). We also provide comparisons with other existing methods in Section 4.3, where we adopt Matthews correlation for CoLA, Spearman correlation for STS-B, and accuracy for the rest ones (i.e., RTE, MRPC, SST-2, QQP, MNLI). We also report the averaged performance on GLUE as an overview.

Implementation. We use the standardly fine-tuned BERT-base and BERT-large models on downstream tasks for both quantization-aware training and post-training quantization. We implement MREM in both the sequential training (abbrv. MREM-S) in Section 3.1 and parallel training with teaching forcing (abbrv. MREM-P) in Section 3.3. For each module, we train for 2,000 steps with an initial learning rate of 1e-4 on GLUE tasks, and 4,000 steps with an initial learning rate of 5e-5 on SQuAD datasets. The learning rate decays linearly as done in [24, 50]. By default, we partition the model into 4 modules on 4 NVIDIA-V100 GPUs. For baselines, we mainly compare with QAT and REM, where the former measures how far is PTQ from QAT, and the latter studies the effect of module-wise granularity in PTQ training. For a fair comparison of each method, we use the same quantization scheme for all methods, i.e., TWN or LAQ for 2-bit and 4-bit weight quantization, and LSQ for all activation quantization. Unlike QAT that picks the best model based on the development set results, MREM is only tested once after training, which does not require further access to the development set. We leave more details of baseline implementation in Appendix B.1. The comparison with more published quantization baselines are left in Section 4.3.

4.2 Main Results: Comparison with QAT and REM

We first compare MREM-S and MREM-P with QAT and REM over MNLI and SQuAD benchmarks. We take BERT-base and BERT-large as backbone PLMs for quantization. The results on MNLI and SQuADv1.1 are summarized in Table 1 and Table 2 respectively, and results on SQuAD v2.0 are left to Table 6 in Appendix B.2. We again summarize according to the four dimensions in Section 2.2.

Performance. It can be found that our proposed MREM-S improves the performance of REM significantly given the same training time, and is much closer to QAT. For instance, according to in Table 1, MREM-S with 4-bit weight quantization on BERT-base and BERT-large achieves accuracies of 83.5%±0.1 and 86.1%±0.1 on the matched section of MNLI, which is on average 10.2%↑ and 16.1%↑ better than REM, and only 1.1%↓ and 0.8%↓ inferior to QAT, respectively. Moreover, with all modules trained in parallel, MREM-P is still close to or only slightly inferior to MREM-S. From Table 2, MREM-P can even outperform MREM-S with the “W2-E2-A4” quantized BERT-large model on SQuAD 1.1, i.e., the EM score and F1 score are on average 0.4%↑ and 0.2%↑ respectively.

Training Time. Our proposed MREM also enjoys significantly less training time than QAT. From Table 1, MREM only takes 84 minutes to quantize BERT-large with 4-bit weights, which is about 38×
Table 1: Results of our proposed MREM-S and MREM-P against QAT and REM on the development set of MNLI. "#Bits (W-E-A)" represents the bit-width for weights of Transformer layers, word embedding, and activations. Acc-m and Acc-mm denote accuracies on the matched and mismatched sections of MNLI respectively.

<table>
<thead>
<tr>
<th>#Bits (W-E-A)</th>
<th>Quant Method</th>
<th>BERT-base</th>
<th>BERT-large</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (min)</td>
<td>Mem (GB)</td>
<td># Data</td>
</tr>
<tr>
<td>full prec.</td>
<td>11.9</td>
<td>84.6</td>
<td>84.9</td>
</tr>
<tr>
<td>2-2-4</td>
<td>4.6</td>
<td>83.9</td>
<td>83.9</td>
</tr>
<tr>
<td>2-2-8</td>
<td>4.4</td>
<td>83.9</td>
<td>83.9</td>
</tr>
<tr>
<td>full prec.</td>
<td>17.7</td>
<td>88.8</td>
<td>88.7</td>
</tr>
<tr>
<td>2-2-4</td>
<td>3.3</td>
<td>89.2</td>
<td>89.2</td>
</tr>
<tr>
<td>2-2-8</td>
<td>3.3</td>
<td>89.2</td>
<td>89.2</td>
</tr>
<tr>
<td>full prec.</td>
<td>33.1</td>
<td>88.8</td>
<td>88.7</td>
</tr>
<tr>
<td>2-2-4</td>
<td>3.3</td>
<td>89.2</td>
<td>89.2</td>
</tr>
<tr>
<td>2-2-8</td>
<td>3.3</td>
<td>89.2</td>
<td>89.2</td>
</tr>
</tbody>
</table>

Table 2: Results of our proposed MREM-S and MREM-P against QAT and REM on the development set of SQuAD v1.1. ‘--’ denotes results with two gradient accumulation steps under the same total batch size due to memory constraint.

<table>
<thead>
<tr>
<th>#Bits (W-E-A)</th>
<th>Quant Method</th>
<th>BERT-base</th>
<th>BERT-large</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (min)</td>
<td>Mem (GB)</td>
<td># Data</td>
</tr>
<tr>
<td>full prec.</td>
<td>11.7</td>
<td>88.8</td>
<td>88.7</td>
</tr>
<tr>
<td>2-2-4</td>
<td>3.3</td>
<td>89.2</td>
<td>89.2</td>
</tr>
<tr>
<td>2-2-8</td>
<td>3.3</td>
<td>89.2</td>
<td>89.2</td>
</tr>
</tbody>
</table>

Table 3: Results of our proposed MREM-S and MREM-P against QAT and REM on the development set of SQuAD v2.1. “--” denotes results with two gradient accumulation steps under the same total batch size due to memory constraint.

<table>
<thead>
<tr>
<th>#Bits (W-E-A)</th>
<th>Quant Method</th>
<th>BERT-base</th>
<th>BERT-large</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (min)</td>
<td>Mem (GB)</td>
<td># Data</td>
</tr>
<tr>
<td>full prec.</td>
<td>11.7</td>
<td>88.8</td>
<td>88.7</td>
</tr>
<tr>
<td>2-2-4</td>
<td>3.3</td>
<td>89.2</td>
<td>89.2</td>
</tr>
<tr>
<td>2-2-8</td>
<td>3.3</td>
<td>89.2</td>
<td>89.2</td>
</tr>
</tbody>
</table>

faster than QAT and 7× faster than full-precision fine-tuning. Note that the full-precision fine-tuning time is listed to conceptually compare how much time is further required for quantization. Comparing with REM, MREM does not cache the output after every matrix multiplication, which admits more iterations given the same amount of time. We discuss this further in Appendix B.3. Moreover, MREM-P is further 4× faster than MREM-S, which achieves the theoretical linear speedup on 4 GPUs. These together bring more than 150× reduction of training time when compared with QAT.

Memory Overhead. Our MREM-S and MREM-P take only around a third of the GPU memory by QAT, and a half of that by the full-precision fine-tuning, e.g., 29.8 GB for QAT and 10.8 GB for MREM-S on BERT-large. Moreover, we also encounter memory overflow (more than 32GB memory of an NVIDIA V100 GPU) for QAT training on SQuAD due to longer sequence length, and we adopt gradient accumulation that inevitably doubles the training time (i.e., underlined figures ‘--’) in Table 2. On the other hand, such issues can be easily mitigated in both REM and our proposed MREM, both of which can be even fed into a single NVIDIA GTX 1080 Ti GPU. We may further decrease the memory overhead of MREM to REM by increasing the number of modules, but this could harm the performance as discussed in Section 4.4.

Data Accessibility. Both REM and our proposed MREM follow the common practice of PTQ, relying only on 4,096 randomly sampled instances on both MNLI and SQuAD, which is a tiny fraction of the original dataset. We shall provide more discussion on the calibration size in Section 4.4.

4.3 Main Results: Comparison with Existing Methods

Next, we compare our MREM with existing state-of-the-art BERT quantization methods. They include various QAT approaches such as Q-BERT [41], Quant-Noise [15], TernaryBERT [56], and
Table 3: Results on the GLUE development set. “Size” refers to model storage in “MB”. “PTQ” indicates whether the method belongs to post-training quantization. “Avg.” denotes the average results of all tasks.

<table>
<thead>
<tr>
<th>Quant Method</th>
<th>#Bits (W-E-A)</th>
<th>Size (MB)</th>
<th>PTQ</th>
<th>MNLI-m</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;W2/4-E8-A4&quot;</td>
<td>full-prec. 488</td>
<td>84.9</td>
<td>91.4</td>
<td>92.1</td>
<td>91.2</td>
<td>59.7</td>
<td>90.1</td>
<td>86.3</td>
<td>72.2</td>
<td>83.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;W2/4-E8-A4&quot;</td>
<td>Q-BERT 2/4-8-8</td>
<td>83.5</td>
<td>-</td>
<td>92.6</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Quant-Noise PQ</td>
<td>2/4-8-8</td>
<td>83.6</td>
<td>-</td>
<td>92.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TermedBERT 2/2-4</td>
<td>83.1</td>
<td>90.1</td>
<td>91.1</td>
<td>92.8</td>
<td>55.7</td>
<td>87.9</td>
<td>87.5</td>
<td>72.9</td>
<td>82.7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOBO 3-4-3</td>
<td>83.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>88.3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GOBO 2-3-2</td>
<td>71.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>82.7</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BRECQ 8-4-4</td>
<td>31.9</td>
<td>62.3</td>
<td>50.7</td>
<td>50.9</td>
<td>0.9</td>
<td>6.4</td>
<td>31.7</td>
<td>52.3</td>
<td>35.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QDrop 8-4-4</td>
<td>71.4</td>
<td>79.0</td>
<td>76.8</td>
<td>88.1</td>
<td>40.9</td>
<td>81.9</td>
<td>79.2</td>
<td>60.7</td>
<td>72.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REM 4-4-2</td>
<td>75.0/0.3</td>
<td>84.9/0.2</td>
<td>86.1/0.2</td>
<td>88.6/0.2</td>
<td>36.3/1.0</td>
<td>82/0.1</td>
<td>81/0.0</td>
<td>67/2.0</td>
<td>75/3.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REM 2-2-2</td>
<td>72.9/0.5</td>
<td>81.5/0.1</td>
<td>84.9/0.2</td>
<td>88.1/0.4</td>
<td>36.2/1.7</td>
<td>80/0.0</td>
<td>81/0.0</td>
<td>65/3.0</td>
<td>73/6.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRECQ 2-2-2</td>
<td>58/3/0.5</td>
<td>75/7/0.3</td>
<td>75/7/0.4</td>
<td>82/9/0.3</td>
<td>16/4/1.2</td>
<td>44/1/4</td>
<td>74/0/0</td>
<td>63/2/0</td>
<td>61/2/0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Ablation studies of teacher forcing at different training steps over MNLI-m.

<table>
<thead>
<tr>
<th>#Bits (W-E-A)</th>
<th># Steps</th>
<th>BERT-base</th>
<th>BERT-large</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>w/o TF</td>
<td>w/ TF</td>
<td>w/o TF</td>
</tr>
<tr>
<td>2-2-8</td>
<td>50 0.2</td>
<td>81.3/0.2</td>
<td>81.3/0.2</td>
</tr>
<tr>
<td>2,000</td>
<td>82.2/0.2</td>
<td>82.2/0.2</td>
<td>84.3/0.1</td>
</tr>
<tr>
<td>4,000</td>
<td>82.3/0.2</td>
<td>82.5/0.2</td>
<td>84.5/0.2</td>
</tr>
</tbody>
</table>

2-2-4 | 50 0.2 | 75.3/0.2 | 77.8/0.2 | 76.5/0.2 | 78.5/0.4 |
| 2,000 | 80.4/0.2 | 80.4/0.2 | 82.9/0.2 | 83.5/0.2 |
| 4,000 | 80.8/0.2 | 81.0/0.2 | 83.1/0.2 | 83.3/0.2 |

Table 5: Comparison of BERT-base with and without per-channel quantization (PCQ) on MNLI.

<table>
<thead>
<tr>
<th>#Bits (W-E-A)</th>
<th>Methods</th>
<th>w/o PCQ</th>
<th>w/o PCQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc  m(%)</td>
<td>Acc  m(%)</td>
<td>Acc  m(%)</td>
</tr>
<tr>
<td>4-4-8</td>
<td>REM</td>
<td>84.3/0.3</td>
<td>77.9/0.2</td>
</tr>
<tr>
<td>MREM</td>
<td>83/0.1</td>
<td>83/0.1</td>
<td>84/0.1</td>
</tr>
</tbody>
</table>

2-2-8 | REM | 71/6.0 | 74/3.0 |
| MREM | 82/7.0 | 82/7.0 | 82/7.0 |

2-2-4 | REM | 59/3.0 | 60/2.0 |
| MREM | 81/1.0 | 81/1.0 | 81/5.0 |

4.4 Discussions

In this section, we provide further discussions to better understand the proposed approach. Unless specified otherwise, all experiments are built upon the BERT-base model over the MNLI dataset.

Teacher Forcing. We now study how teacher forcing benefits MREM-P with different numbers of training steps, and results are listed in Table 4. We find that teacher forcing brings consistent improvement for both BERT-base and BERT-large, and the gain is more significant with fewer training steps or lower quantization bit-width. For example, it brings 3.4% ↑ and 2.8% ↑ on the “W2-E2-A4” quantized BERT-base and BERT-large respectively under 250 steps. This matches our intuition that fewer training steps or higher compression ratio give larger reconstruction error, when the clean input from the full-precision module can benefit more the quantized module. As more training steps bring only marginal improvement, we by default set the training steps to 2, 000.

Additionally, we also plot training loss curves of the 2-nd and 4-th modules under 250 and 2, 000 training steps in Figure C, and more visualizations can be found in Figure D of Appendix B.4. We find that: 1) the loss curves with teacher forcing are apparently lower, especially when trained with fewer steps, which is consistent with Table 4 2) the loss curves rise in the halfway during teacher forcing, since the modules need to adapt to quantized input with accumulated errors; 3) the loss curves of the 4-th modules are lower than the 2-nd ones, which matches the intuition that the late modules have
Figure 4: The training loss curves with and without teacher forcing (TF) in MREM-P. The shaded area denotes teacher forcing in the first 40% training steps. We show the 2-nd and 4-th modules trained with 250 steps and 2,000 steps in (a), (b) and (c), (d) respectively.

Figure 5: Discussions on the proposed MREM approach. In (a) and (b), the solid line and shaded area denote the averaged results and standard deviation of a “W2-E2-A4” quantized BERT-base model from 10 different seeds. (c) and (d) visualize the propagation of reconstruction error on “W2-E2-A8” and “W2-E2-A4” quantized BERT-base model, respectively.

more errors accumulated and thus benefit more from teacher forcing. We also try tuning the teacher forcing steps (i.e., shaded area) within [20%, 80%] of total iterations, but observe no large difference in performance. We thus choose 40% training steps for teacher forcing by default.

Number of Modules and Memory Overhead. We verify the effect of model partition on the final quantized performance, as well as their corresponding memory consumption. According to Figure 5(a) by varying the number of modules within {1, 2, 3, 4, 6}, it can be found that more model partitions give slightly lower performance, as layer-wise dependencies are less considered during reconstruction error minimization. However, more partitions lead to less running memory, i.e., {11.9, 6.7, 4.7, 3.7, 2.7} GB for these partitions correspondingly. As the decrease of memory diminishes with more partitions, we partition the model into 4 modules by default.

Size of Calibration Data. The size of calibration data directly relates to the data accessibility issue in post-training quantization. To learn its effects, we vary the calibration data size $|\tilde{D}|$ within {32, 64, 128, 512, 1024, 2048, 4096, 8192}, and list the results of REM, MREM-S and MREM-P. From Figure 5(b), it can be found that while REM is ahead of MREM-S/P with fewer than 128 training samples, the accuracy of REM rises slowly and saturates at around 60% afterwards. We hypothesize that the simple training objective in REM can hardly hold more training instances for optimization. MREM-S/P, on the other hand, can better exploit larger calibration data size, since the module-wise granularity admits higher flexibility for the optimization. As we find the diminishing gain to increase the training size after 4.096 samples, we by default take 4.096 samples.

Reconstruction Error Propagation. We visualize the propagation of reconstruction error for both “W2-E2-A8” and “W2-E2-A4” quantized BERT-base models in Figure 5(c) and Figure 5(d) respectively. It can be observed that our MREM achieves both lower values and slower rising rates of the reconstruction error than REM across all layers, which verifies the advantage of module-wise granularity to minimize the reconstruction error. Interestingly, while the reconstruction error generally gets enlarged layer-wisely in the first ten layers, it begins to decrease afterwards. We speculate this is due to the classification head that encourages concentrated hidden representations for the task.
Per-channel Quantization. Per-channel Quantization (PCQ) is prevalent in the post-training quantization of convolution neural networks \cite{35,33,23}. To learn its effect in PLMs, PCQ assigns different quantization step-sizes at each output dimension of the linear layer, which is also known as row-wise quantization in \cite{56}. The PCQ results of REM and MREM are shown in Table \ref{table:rem_mrem}. It can be found that while PCQ improves REM by around 1.0% to 2.5%, the gain is very incremental on MREM. Our results are also similar to the findings in \cite{56}, where the row-wise quantization brings little improvement. As PCQ requires to store more full-precision step sizes with minor improvement, we do not employ PCQ by default.

5 Related Work

Network quantization is a common approach to compress and accelerate deep neural networks \cite{8,26,20,19,13,28,7}. For network quantization of pre-trained language models, previous efforts mostly adopt quantization-aware training \cite{55,41,56,5,38}. For instance, Q8BERT \cite{55} converts both parameters and activations with 8-bit representations with negligible task degradation. Q-BERT \cite{41} exploits the hessian matrix of loss curvature to determine the layer-wise quantization bit-width, achieving a higher compression rate. TernaryBERT \cite{56} proposes to ternarize BERT parameters with 2-bit representations, while BinaryBERT \cite{3} further uses 1-bit to represent model parameters. BiBERT \cite{38} explores the fully-binarized BERT with 1-bit weights and activations. Despite the success by these efforts, the heavy fine-tuning of QAT makes it prohibitive given constraints on training time, memory size, and data accessibility.

Post-training quantization \cite{17,53,11} is thus considered mitigate these issues from QAT. One line of PTQ research quantizes the network purely without using any training data, but removes outliers in the full-precision parameters \cite{57,54,16,54}. Methodologically, PTQ can be achieved by splitting an outlier neuron with a large magnitude into two parts \cite{57}, where the magnitude can be halved. Alternatively, one can scale down outlier magnitude and multiply it back in subsequent layers, a.k.a. weight equalization in \cite{53}. Another solution is to treat the outliers and normal values in the distribution separately, by keeping two sets of quantization parameters \cite{16,54}. Another line in PTQ research \cite{33,57,33,23} minimizes the reconstruction error based on the calibration data. Compared with training-free PTQ approaches, such an approach significantly improves the performance of the quantized network, and we thus follow this line in this paper.

Recently, there are also concurrent efforts with us to apply PTQ for PTMs \cite{49,57,53,11}. In \cite{49}, a similar method is developed to minimize the quantization error via the calibration data. Moreover, PTQ is also explored for large generative PTMs (e.g. GPT-3 \cite{5}) from 125M to 175B \cite{57,53,11}.

6 Conclusion

In this paper, we study post-training quantization for pre-trained language models. We show that existing quantization-aware training solutions suffer from slow training, huge memory overhead, and data privacy issues when accessing the full training set. To mitigate these issues, we propose module-wise reconstruction error minimization, an efficient solution to quantize PLMs. MREM can be conducted either sequentially or in parallel, where the parallel training can achieve the speedup close to the theoretical limit without apparent performance degradation. Experimental results show that the proposed solution greatly improves the performance. Meanwhile, it significantly reduces the training time and memory overhead with only thousands of training instances. We leave the broader impact and limitations of this work in Appendix \ref{appendix:impact_and_limitations}.

Acknowledgement and Disclosure of Funding

We gratefully acknowledge the support of MindSpore for this research, as well as the insightful suggestions from the anonymous reviewers. The work described in this paper was partially supported by the National Key Research and Development Program of China (No. 2018AAA0100204), the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 241021 of the Research Impact Fund, No. R5034-18; and No. CUHK 14210920 of the General Research Fund).
References


Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default [TODO] to [Yes], [No], or [N/A]. You are strongly encouraged to include a justification to your answer, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section ??.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] In the code document.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you’re using/curating? [No] Their license permits public use for research purpose.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] GLUE and SQuAD are widely used benchmark dataset in NLP research.

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]