Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT

Sheng Shen,[1*] Zhen Dong,[1*] Jiayu Ye,[1*] Linjian Ma,[†1] Zhewei Yao,[1]
Amir Gholami,[1] Michael W. Mahoney,[1] Kurt Keutzer[1]
[1]University of California at Berkeley,
{sheng.s, zhendong, yejiayu, linjian, zheweiy, amirgh, mahoneymw, keutzer}@berkeley.edu

Abstract

Transformer based architectures have become de-facto models used for a range of Natural Language Processing tasks. In particular, the BERT based models achieved significant accuracy gain for GLUE tasks, CoNLL-03 and SQuAD. However, BERT based models have a prohibitive memory footprint and latency. As a result, deploying BERT based models in resource constrained environments has become a challenging task. In this work, we perform an extensive analysis of fine-tuned BERT models using second order Hessian information, and we use our results to propose a novel method for quantizing BERT models to ultra low precision. In particular, we propose a new group-wise quantization scheme, and we use Hessian-based mix-precision method to compress the model further. We extensively test our proposed method on BERT downstream tasks of SST-2, MNLI, CoNLL-03, and SQuAD. We can achieve comparable performance to baseline with at most 2.3% performance degradation, even with ultra-low precision quantization down to 2 bits, corresponding up to $13\times$ compression of the model parameters, and up to $4\times$ compression of the embedding table as well as activations. Among all tasks, we observed the highest performance loss for BERT fine-tuned on SQuAD. By probing into the Hessian based analysis as well as visualization, we show that this is related to the fact that current training/fine-tuning strategy of BERT does not converge for SQuAD.

1 Related Work

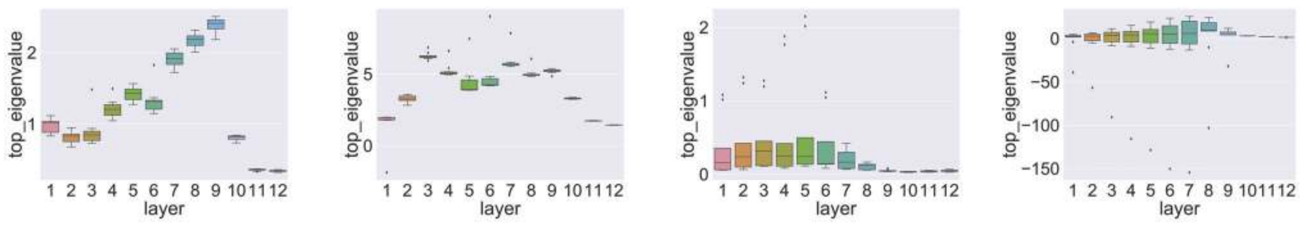
Model compression Model compression is a very active area of research. Efforts in this area could be broadly categorized as follows: (i) new architectures that are compact by design (Iandola et al. 2016; Howard et al. 2017); (ii) automated neural architecture search (NAS) with reward function set as latency or model size (Wang et al. 2019); (iii) pruning based methods to reduce model size of existing architectures (LeCun, Denker, and Solla 1990; Hassibi, Stork, and Wolff 1994); (iv) knowledge distillation from a large model to help train a more compact model (Ba and Caruana 2014; Hinton, Vinyals, and Dean 2015); (v) hardware and architecture co-design (Gholami et al. 2018); and (vi) inference quantization (Zhang, Choromanska, and LeCun 2015; Dong et al. 2019).

[*]Equal contribution.

[†]Work done while interning at Wave Computing.

Here we solely focus on quantization (Courbariaux, Bengio, and David 2015; Rastegari et al. 2016; Li, Zhang, and Liu 2016; Zhou et al. 2016; Choi et al. 2018; Dong et al. 2019; Zhang et al. 2018). One of the challenges here is that ultra low precision quantization can lead to significant accuracy degradation. Mixed precision quantization (Wu et al. 2018; Zhou et al. 2018; Wang et al. 2019) and multi-stage quantization (Zhou et al. 2017) have been proposed to solve/alleviate this problem. However, the challenge with mixed-precision quantization is that the search space is exponentially large. For instance, if we have three precision options for a specific layer (2, 4 or 8-bits), then the total search space of each fine-tuned BERT model (Devlin et al. 2019) becomes $3^{12} \approx 5.3\times 10^5$ different precision settings. Recently, (Dong et al. 2019) proposed a second-order sensitivity based method to address this issue and achieved state-of-the-art results on computer vision tasks. Part of our paper builds upon this prior work and extends the results to include other variations of second order information instead of just the mean value of the Hessian spectrum.

Compressed NLP model Notable examples for NLP compression work are LSTM and GRU-based models for machine translation and language model (Xu et al. 2018; Wang et al. 2018). From the recent introduction of Tranformer models, we have observed a significant increase in NLP model size. This is due to the incorporation of very large fully connected layers and attention matrices in Transformers (Vaswani et al. 2017; Devlin et al. 2019; Yang et al. 2019; Liu et al. 2019; Radford et al. 2019). Model compression is crucial for deploying these models in resource constrained environments. Pilot work addressing this are (Michel, Levy, and Neubig 2019; Bhandare et al. 2019). From a different angle, (Tay et al. 2019; Ma et al. 2019) have probed the architectural change of self-attention layer to make the Transformer lightweight. There have also been attempts to use distillation to reduce large pre-trained Transformer models such as BERT (Devlin et al. 2019) in (Tang et al. 2019; Sun et al. 2019). However, significant accuracy loss is observed even for relatively small compression ratio of $4\times$. Here we show that this compression ratio could be increased up to $13\times$, including $4\times$ reduction of embedding layer, with much smaller performance degradation.



(a) SST-2 (b) MNLI (c) CoNLL-03 (d) SQuAD

Figure 1: From (a) to (d): Top eigenvalue distributions for different encoder layers for SST-2, MNLI, CoNNL-03, SQuAD, respectively. Layers in the middle have higher mean values and larger variance than the others. The last three layers have the smallest variance and mean values among all layers.

2 Methodology

In this section, we introduce our proposed BERT quantization methods, including the mixed precision quantization based on Hessian information, as well as techniques used for the group-wise quantizing scheme.

As in (Devlin et al. 2019), a fine-tuned BERTBASE model consists of three parts: embedding; Transformer based encoder layers; and output layer. Specifically, assuming $x \in X$ is the input word (sentence) and $y \in Y$ is the corresponding label, we have the loss function L defined as:

$$L(\theta) = \sum_{(x_i, y_i)} \text{CE}(\text{softmax}(W_c(W_n(...W_1(W_e(x_i))))), y_i),$$

where CE is the cross entropy function (or other appropriate loss functions), θ is a combination of $W_e, W_1, W_2, ..., W_n$ and W_c. Here, W_e is the embedding table, $W_1, W_2, ..., W_n$ are the encoder layers, and W_c is the output/classifier layer[1].

The size of parameters in BERTBASE model is 91MB for embedding, 325MB for encoder and 0.01MB for output. We do not quantize the output layer due to its negligible size, and focus on quantizing both the embedding and encoder layers. As will be discussed in Sec. 4.1, we find that the embedding layer is much more sensitive to quantization than the encoder layers. As a result, we quantize embedding and encoder parameters in different ways. The quantization schemes we used are explained in detail in the following sections.

2.1 Quantization process

General NN inference is performed in floating point precision for both weights and activations. Quantization restricts the network weights to a finite set of values as $Q(z) = q_j, \quad \text{for } t_j \leq z \leq t_{j+1}$. Here Q is quantization operator, z is a real valued input tensor (activation or a weight). Here k is the quantization precision for a specific layer.

There are multiple choices for quantization function Q. Here we use uniform quantization function, where the range of floating point values in a tensor is equally split (Zhou et al. 2016; Hubara et al. 2017) and then represented by unsigned integers in $\{0, \ldots, 2^k - 1\}$. It should be noted that a non-uniform quantizer can potentially further increase the accuracy. However, we solely focus on uniform quantization since it allows more efficient and easier hardware implementation. To backpropogate gradients through Q, which is non-differentiable, we use the Straight-through Estimator (STE) (Bengio, Léonard, and Courville 2013). See Appendix for more details about the forward and backward propagation during the entire quantization process.

[1]Here, we use W_* for both function and its corresponding parameters without confusion.

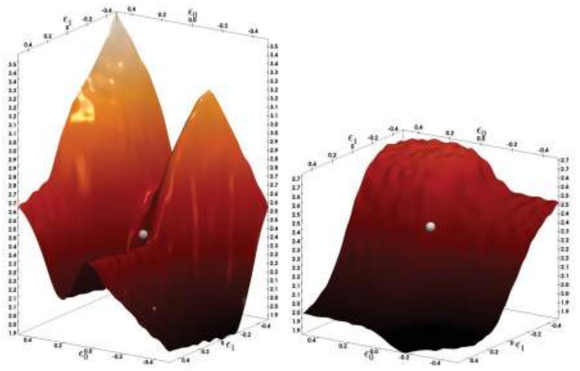
2.2 Mixed precision quantization

Different encoder layers are attending to different structures (Clark et al. 2019), and it is expected that they exhibit different sensitivity. Thus, assigning the same number of bits to all the layers is sub-optimal. This scenario is more critical if the targeted model size is very small, which requires ultra low precision such as 4-bits or 2-bits. As a result we explore mixed-precision quantization, where we assign more bits to more sensitive layers in order to retain performance.

In (Dong et al. 2019), a Hessian AWare Quantization (HAWQ) is developed for mixed-bits assignments. The main idea is that the parameters in NN layers with higher Hessian spectrum (i.e., larger top eigenvalues) are more sensitive to quantization and require higher precision as compared to layers with small Hessian spectrum. However, there exist 7M parameters for each encoder layer in BERTBASE. Given that the Hessian of each layer is a matrix of size $7M \times 7M$, there is a common misconception that computing second order statistics is infeasible. However, the Hessian spectrum can be computed by a matrix-free power iteration method (Yao et al. 2018), which does not require explicit formation of the operator. To illustrate this, we take the first encoder layer as an example. Denoting the gradient of the first encoder layer as g_1, for a random vector v independent with g_1, we have

$$\frac{\partial g_1^T v}{\partial W_1} = \frac{\partial g_1^T}{\partial W_1} v + g_1^T \frac{\partial v}{\partial W_1} = \frac{\partial g_1^T}{\partial W_1} v = H_1 v, \qquad (1)$$

where H_1 is Hessian matrix of the first encoder. The top eigenvalue then can be computed by power iteration, as shown in Appendix. We denote λ_i as the top eigenvalue of i-th encoder layer. Using this approach, we show the distribution of top Hessian eigenvalue for different layers of BERTBASE are shown in Fig. 1. Different layers exhibit different magni-



(a) SQuAD 7th layer (b) SQuAD 11th layer

Figure 2: The loss landscape for different layers in SQuAD is illustrated by perturbing the parameters along the first two dominant eigenvectors of the Hessian. The silver sphere shows the point in the parameter space to which the BERT model has converged.

tude of eigenvalues even though all layers have exactly same structure and size.

The above Hessian based approach was used in (Dong et al. 2019), where top eigenvalues are computed and averaged for different training data. More aggressive quantization is performed for layers that have smaller top eigenvalue, which corresponds to flatter loss landscape as in Appendix. However, we find that assigning bits based only on the average top eigenvalues is infeasible for many NLP tasks. As shown in Fig. 1, top eigenvalues of Hessian for some layers exhibits very high variance with respect to different portion of the input dataset. As an example, the variance of the 7th layer for SQuAD stays larger than 61.6 while the mean of that layer is around 1.0, even though each data point corresponds to 10% of the entire dataset (which is 9K samples). To address this, we use the following metric instead of just using mean value,

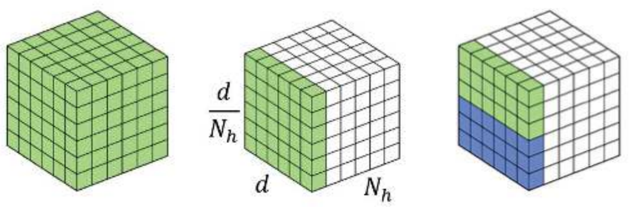
$$\Omega_i \triangleq |\text{mean}(\lambda_i)| + \text{std}(\lambda_i), \tag{2}$$

where λ_i is the distribution of the top eigenvalues of H_i, calculated with 10% of training dataset.[2]

After Ω_i is computed, we sort them in descending order, and we use it as a metric to relatively determine the quantization precision. We then perform quantization-aware finetuning based on the selected precision setting.

An important technical point that we need to emphasize is that our method expects that before performing quantization the trained model has converged to a local minima. That is, the practitioners who trained BERT and performed its fine-tuning for downstream tasks should have chosen the hyper-parameters and number of iterations such that a local minima has been reached. The necessary optimality conditions are zero gradient, and positive curvature (i.e., positive Hessian eigenvalue). In our analysis, we observed that for the three tasks of MNLI, CoNLL-03, and SST-2 the top Hessian eigenvalue is indeed positive for (see Appendix). However,

[2]Without confusion, we use λ_i for both single top eigenvalue and its distribution with respect to 10% of the data.



(a) Layer-wise (b) Group-wise (N_h group) (c) Group-wise ($2N_h$ group)

Figure 3: The overview of Group-wise Quantization Method. We illustrate this with value matrices of a multi-head self attention layer. Here N_h(number of heads) value matrices W_v are concatenated together, which results in a 3-d tensor. The same color denotes the same group with a shared quantization range.

we find that the BERT model fine-tuned for SQuAD has actually *not* converged to a local minima, as evident in the Hessian eigenvalues shown in Fig. 1(d), where we observe very large negative eigenvalues. Directly visualizing the loss landscape also shows this very clearly as in Fig. 2. Because of this, our expectation is that performing quantization on SQuAD would lead to higher performance degradation as compared to other tasks, and this is indeed the case as will be discussed next.

2.3 Group-wise Quantization

Assume that the input sequence has n words and each word has a d-dim embedding vector ($d = 768$ for BERTBASE), i.e., $x = (x(1), \ldots, x(n))^T \in \mathbb{R}^{n\times d}$. In Transformer encoder, each self-attention head has 4 dense matrix, i.e., $W_k, W_q, W_v, W_o \in \mathbb{R}^{\frac{d}{N_h}\times d}$, where N_h is the number of attention heads. Here W_k, W_q, W_v and W_o stand for key, query, value and output weight matrix. Each self-attention head computes the weighted sum as

$$\text{Att}(x, x(j)) = W_o \sum_{i=1}^{n} \text{softmax}\left(\frac{x(j)^T W_q^T W_k x(i)}{\sqrt{d}}\right) W_v x(i).$$

Through this reparametrization, the multi-head self-attention (MHSA) will add these features into the final output, that is we will have $\sum_{i=1}^{N_h} \text{Att}_i(x, x(j))$. Directly quantizing each 4 matrices in MHSA as an entirety with the same quantization range can significantly degrade the accuracy, since there are more than 2M parameters in total, which corresponds to $4 \times 12 \times 64 = 3072$ neurons, and the weights corresponding to each neuron may lie in different range of real numbers. Channel-wise quantization can be used to alleviate this problem in convolutional neural networks, where each convolutional kernel can be treated as a single output channel and have its own quantization range. However, this cannot be directly applied for dense matrices, since each dense matrix itself is a single kernel. Therefore, we propose group-wise quantization for attention-based models. We treat the individual matrix W with respect to each head in one dense matrix of MHSA as a group so there will be 12 groups. Furthermore, in each group, we bucket sequential output neurons together as sub-groups, e.g., each 6 output neurons as one sub-group so there are $12 \times \frac{64}{6} = 128$ sub-group in total (the hidden dim in each head of BERTBASE is $\frac{768}{12} = 64$). Each sub-group can have its own quantization range. An illustration is shown in Fig. 3 for W_v, where we concatenate N_h value matrix W_v to be a 3-d tensor. For layer-wise quantization, the entire 3-d tensor will be quantized into the same range of discrete numbers, as shown in Fig. 3a. A special case of group-wise quantization is that we treat each dense matrix as a group, and every matrix can have its own quantization range as shown in Fig. 3b. A more general case in Fig. 3c is that we partition each dense matrix with respect to output neuron, and we bucket every continuous $\frac{d}{2N_h}$ output neurons as a group. The effect of finer group-wise quantization is further investigated in Sec. 3.2.

3 Experiment

In this section, we describe our experiments on evaluating the proposed Q-BERT on four different NLP tasks. Details of the datasets are shown in Appendix. To the best of our knowledge, there is no published work done on BERT quantization at this point, so we report Direct quantization (DirectQ), i.e., quantization without mixed-precision and group-wise quantization as a baseline.

3.1 Main Results

We present results of Q-BERT on the development set of the four tasks of SST-2, MNLI, CoNLL-03, and SQuAD, as summarized in Tab. 1 and 2. As one can see, Q-BERT performs significantly better compared to the DirectQ method across all the four tasks in each bit setting. The gap becomes more obvious for ultra low bit setting. As an example, in 4-bits setting, Direct quantization (DirectQ) of SQuAD results in 11.5% performance degradation as compared to BERTBASE. However, for the same 4-bits setting, Q-BERT only exhibits 0.5% performance degradation. Moreover, under 3-bits setting, the gap between Q-BERT and DirectQ increases even further to 9.68-27.83% for various tasks.

In order to push further the precision setting to lower bits, we investigate the mixed-precision Q-BERT (Q-BERTMP). As can be seen, Q-BERT with uniform 2-bits setting has very poor performance across all four tasks, though the memory is reduced by 20% against 3-bits setting. The reason behind this is the discrepancy that not all the layers have the same sensitivity to quantization as evident from loss landscape visualizations; see Appendix. Intuitively, for more sensitive layers, higher bit precision needs to be set, while for layers that are less sensitive, 2-bits setting is already sufficient. To set mixed precision to each encoder layer of BERTBASE, we measure the sensitivity based on Eq. 2, which captures both mean and variance of the top eigenvalue of the Hessian shown in Fig. 1. Note that all experiments in Fig. 1 are based on 10 runs and each run uses 10% of the entire training dataset. We can obverse that for most of the lower encoder layers (layer 1-8), the variance is pretty large compared to the last three layers. We generally observe that the middle part (layer 4-8)

Table 1: Quantization results for BERTBASE on Natural Language Understanding tasks. Results are obtained with 128 groups in each layer. We abbreviate quantization bits used for weights as "w-bits", embedding as "e-bits", model size in MB as "Size", and model size without embedding layer in MB as "Size-w/o-e". For simplicity and efficacy, all the models except for Baseline are using 8-bits activation. Furthermore, we compare Q-BERT with direct quantization method ("DirectQ") without using mixed precision or group-wise quantization. Here "MP" refers to mixed-precision quantization.

(a) SST-2

Method	w-bits	e-bits	Acc	Size	Size-w/o-e
Baseline	32	32	93.00	415.4	324.5
Q-BERT	8	8	92.88	103.9	81.2
DirectQ	4	8	85.67	63.4	40.6
Q-BERT	4	8	**92.66**	63.4	40.6
DirectQ	3	8	82.86	53.2	30.5
Q-BERT	3	8	**92.54**	53.2	30.5
Q-BERTMP	2/4 MP	8	**92.55**	53.2	30.5
DirectQ	2	8	80.62	43.1	20.4
Q-BERT	2	8	**84.63**	43.1	20.4
Q-BERTMP	2/3 MP	8	**92.08**	**48.1**	**25.4**

(b) MNLI

Method	w-bits	e-bits	Acc m	Acc mm	Size	Size w/o-e
Baseline	32	32	84.00	84.40	415.4	324.5
Q-BERT	8	8	83.91	83.83	103.9	81.2
DirectQ	4	8	76.69	77.00	63.4	40.6
Q-BERT	4	8	**83.89**	**84.17**	63.4	40.6
DirectQ	3	8	70.27	70.89	53.2	30.5
Q-BERT	3	8	**83.41**	**83.83**	53.2	30.5
Q-BERTMP	2/4 MP	8	**83.51**	**83.55**	53.2	30.5
DirectQ	2	8	53.29	53.32	43.1	20.4
Q-BERT	2	8	**76.56**	**77.02**	43.1	20.4
Q-BERTMP	2/3 MP	8	**81.75**	**82.29**	**46.1**	**23.4**

(c) CoNLL-03

Method	w-bits	e-bits	F_1	Size	Size-w/o-e
Baseline	32	32	95.00	410.9	324.5
Q-BERT	8	8	94.79	102.8	81.2
DirectQ	4	8	89.86	62.2	40.6
Q-BERT	4	8	**94.90**	62.2	40.6
DirectQ	3	8	84.92	52.1	30.5
Q-BERT	3	8	**94.78**	52.1	30.5
Q-BERTMP	2/4 MP	8	**94.55**	52.1	30.5
DirectQ	2	8	54.50	42.0	20.4
Q-BERT	2	8	**91.06**	42.0	20.4
Q-BERTMP	2/3 MP	8	**94.37**	**45.0**	**23.4**

Table 2: Quantization results for BERTBASE on SQuAD.

Method	w-bits	e-bits	EM	F_1	Size	Size-w/o-e
Baseline	32	32	81.54	88.69	415.4	324.5
Q-BERT	8	8	81.07	88.47	103.9	81.2
DirectQ	4	8	66.05	77.10	63.4	40.6
Q-BERT	4	8	**80.95**	**88.36**	63.4	40.6
DirectQ	3	8	46.77	59.83	53.2	30.5
Q-BERT	3	8	**79.96**	**87.66**	53.2	30.5
Q-BERTMP	2/4 MP	8	**79.85**	**87.49**	53.2	30.5
DirectQ	2	8	4.77	10.32	43.1	20.4
Q-BERT	2	8	**69.68**	**79.60**	43.1	20.4
Q-BERTMP	2/3 MP	8	**79.25**	**86.95**	**48.1**	**25.4**

has the largest mean(λ_i). Beyond the relatively smaller mean, the last three layers also have much smaller variance, which indicates the insensitivity of these layers. Therefore, higher bits will only be assigned for middle layers according to Eq. 2 for Q-BERT 2/3 MP.[3] In this way, with only additional 5MB memory storage, 2/3-bits Q-BERTMP is able to retain the performance drop within 2.3% for MNLI, SQuAD and 1.1% for SST-2, CoNLL-03, with up to $13\times$ compression ratio in weights. Note that this is up to 6.8% better than Q-BERT with uniform 2 bits.

One consideration for quantization is that 3-bit quantized execution is typically not supported in hardware. It is however possible to load 3-bit quantized values and cast them to higher bit precision such as 4 or 8 bits in the execution units. This would still have the benefit of reduced memory volume to/from DRAM. It is also possible to avoid using 3 bits and instead use a mixture of 2 and 4 bits as shown in Tab. 1. For example, SST-2 Q-BERTMP with mixed 2/4-bit precision weights has the same model size as the 3 bit quantization in 53.2MB and achieves similar accuracy. We observe similar trend for other tasks as well.

One important observation is that we found SQuAD to be harder to quantize as compared to other tasks; see Tab. 2. For example, 2-bits DirectQ results in more than 10% F_1 score degradation. Even Q-BERT has larger performance drop as compared to other tasks in Tab. 1. We studied this phenomenon further through Hessian analysis. In Fig. 1, among all the tasks, it can be clearly seen that SQuAD not only has much larger eigenvalue variance, but it has very large negative eigenvalues. In fact this shows that the existing BERT model for SQuAD has not reached a local minima. This is further illustrated in the 3-d loss landscape of all four tasks in Appendix. It can be clearly seen that for other three tasks, the stopping point is at a quadratic bowl (at least in the first two dominant eigenvalue directions of the Hessian). However, compared to the others, SQuAD has a totally different structure to its loss landscape. As shown in Fig. 2, the stopping points of different layers on SQuAD have negative curvature directions, which means they have not converged to a local minima yet. This could well explain why the quantization of SQuAD results in more accuracy drop. Our initial attempts to address this by changing training hyper-parameters were not successful. We found that the BERT model quickly overfits the training data. However, we emphasize that fixing BERT model training itself is outside the scope of this paper and not possible with academic computational resources.

3.2 Effects of group-wise quantization

We measure the performance gains with different group numbers in Tab. 3. We can observe from the table that performing layer-wise quantization (shown in Fig. 3a) is sub-optimal for all four tasks (the performance drop is around 7% to 11.5%). However, the performance significantly increases as we increase the number of groups. For example, for 12 groups, the performance degradation is less than 2% for all the tasks. Further increasing the group number from 12 to 128 increases the accuracy further by at least 0.3% accuracy. However, increasing the group number further from 128 to 768 can only increase the performance within 0.1%. This shows that the performance gain almost saturates around 128 groups. It is also preferable not to have very large value for the number of group since it increases the number of Look-up Tables (LUTs) necessary for each matrix multiplication which can adversely affect hardware performance, and based on our results there are diminishing returns in terms of accuracy. In all our experiments, we used 128 groups for both Q-BERT and Q-BERTMP in Sec. 3.1.

Table 3: Effects of group-wise quantization for Q-BERT on three tasks. The quantization bits were set to be 4 for weights, 8 for embeddings and 8 for activations for all the tasks.

# Group	SST-2	MNLI-m/mm	CoNLL-03
Baseline	93.00	84.00/84.40	95.00
1	85.67	76.69/77.00	89.86
12	92.31	82.37/82.95	94.42
128	92.66	83.89/84.17	94.90
768 [4]	92.78	84.00/84.20	94.99

4 Discussion

In this Section, we further investigate the quantization effects on different modules, e.g. different embedding layers (e.g., word and position embeddings), and we perform qualitative analysis using attention distribution. This illustrates that Q-BERT better captures the behaviour of the original model as compared to DirectQ in all cases.

4.1 Quantization effects on different modules

Here we investigate the quantization effects with respect to different modules of BERT model (multi-head self-attention versus feed-forward network, and different embedding layers, i.e., word embedding versus position embedding).

Generally speaking, we find that embedding layer is more sensitive than weights for quantization. This is illustrated in Tab. 4a, where we use 4-bits layerwise quantization for

[3]Exact detailed bits setting is included in the Appendix

[4]Here we treat each output neuron as a single group.

embedding, which results in an unacceptable performance drop up to 10% for SST-2, MNLI, CoNLL-03 and even more than 20% for SQuAD. This is despite the fact that we used 8/8-bits for weights/activations. On the contrary, encoder layers consume around 79% total parameters ($4\times$ embedding parameter size), while quantizing them to 4-bits in Tab. 1 and 2 leads to less performance loss.

Furthermore, we find that position embedding is very sensitive to quantization. For instance, quantizing position embedding to 4 bits results in generally 2% additional performance degradation than quantizing word embedding, even though the position embedding only accounts for less than 5% of the entire embedding. This indicates the importance of positional information in Natural Language Understanding tasks. Given position embedding only accounts for a small portion of model size, we can do mixed-precision quantization for embedding to further push down the model size boundary with a tolerable accuracy drop, as shown in Appendix.

Table 4: Quantization effect to different modules. We abbreviate the quantization bits used for word embedding as "ew-bits", position embedding as "ep-bits", multi-head attention layer as "s-bits" and fully-connected layer as "f-bits". In (a), we set weight and activation bits as 8. In (b), we set embedding and activation bits as 8.

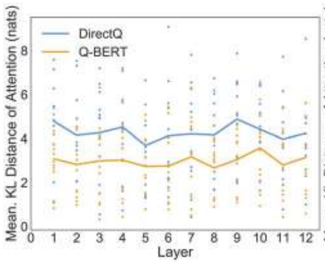
(a) quantization effect on embedding

Method	ew-bits	ep-bits	SST-2	MNLI-m	CoNLL-03	SQuAD
Baseline	32	32	93.00	84.00	95.00	88.69
Q-BERT	8	8	92.88	83.83	94.79	88.47
Q-BERT	4	8	91.74	82.91	94.44	87.55
Q-BERT	8	4	89.11	82.84	93.86	72.38
Q-BERT	4	4	85.55	78.08	84.32	61.70

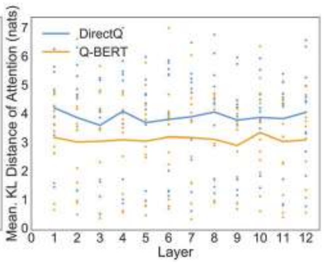
(b) quantization of multi-head attention versus fully-connected layer

Method	s-bits	f-bits	SST-2	MNLI-m	CoNLL-03	SQuAD
Baseline	32	32	93.00	84.00	95.00	88.69
Q-BERTMP	1/2MP	2/3MP	89.56	73.66	91.74	75.81
Q-BERTMP	2/3MP	1/2MP	85.89	70.89	87.55	68.71
Q-BERTMP	2/3MP	2/3MP	92.08	81.75	93.91	86.95

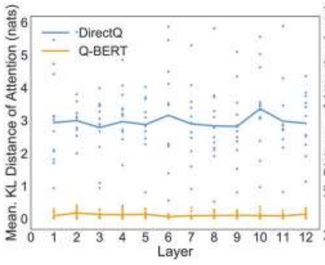
To study the quantization effects on self-attention layers and fully-connected networks, we conducted extensive experiments under different bits settings for the encoder layers. The results are shown in Tab. 4b. Specifically, we adopt the Q-BERTMP setting in Tab. 1, with a mixture of 2 and 3 bits for encoder weights. To test the robustness of the two modules inside each encoder layer, we further reduce one more bit in the corresponding modules and denote the resulting precision setting 1/2MP. From Tab. 4b, we can conclude that generally self-attention layer is more robust to quantization than the fully-connected network, since 1/2MP self-attention results in about 5% performance drop while 1/2MP fully-connected will worsen this to 11%.



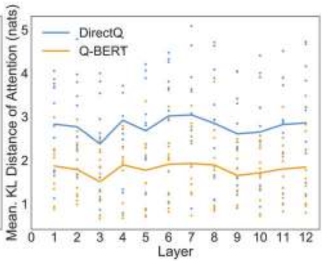
(a) SST-2



(b) MNLI



(c) CoNLL-03



(d) SQuAD

Figure 4: KL divergence over attention distribution between Q-BERT/DirectQ and Baseline. The distance between Q-BERT and Baseline is much smaller than that of DirectQ and Baseline.

4.2 Qualitative Analysis

We use attention information to conduct qualitative analysis to analyze the difference between Q-BERT and DirectQ.

To do so, we compute the Kullback–Leibler (KL) divergence between the attention distribution for the same input from the coordinated head of both quantized BERT and full-precision BERT. It should be noted that we compute the average distance out of 10% of the entire training dataset. The smaller KL divergence here means that the output of the multi-head attention of the two models is closer to each other. We illustrate this distance score for each individual head in Fig. 4 for SST-2, MNLI, CoNLL-03 and SQuAD. We compared Q-BERT and DirectQ with 4-bits weights, 8-bits embedding and 8-bits activation. Each scatter point in Fig. 4 denotes the distance w.r.t one head, and the line chart shows the average results over the 12 heads in one layer. We can clearly see that Q-BERT always incurs a smaller distance to the original baseline model as compared to DirectQ model, for all the different layers.

5 Conclusion

In this work, we perform an extensive analysis of fine-tuned BERT and propose Q-BERT, an effective scheme for quantizing BERT. In order to aggressively reduce the model size by mixed-precision quantization, we proposed a new layer-wise Hessian based method which captures both the average and the variance of the eigenvalues. Moreover, a new group-wise quantization is proposed to perform fine-grained quantization inside each encoder layer. In four downstream tasks, equipped with the aforementioned methods, Q-BERT achieves $13\times$ compression ratio in weights, $4\times$ smaller activation size, and $4\times$ smaller embedding size, with at most 2.3% accuracy loss. To better understand how different factors will affect the trade-off between performance and the model compression ratio in Q-BERT, we conduct controlled experiments to investigate the effect of different quantization schemes and quantizing different modules in BERT, respectively.

References

Ba, J., and Caruana, R. 2014. Do deep nets really need to be deep? In *Proceedings of the NIPS*, 2654–2662.

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

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

Choi, J.; Wang, Z.; Venkataramani, S.; Chuang, P. I.-J.; Srinivasan, V.; and Gopalakrishnan, K. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv:1805.06085*.

Clark, K.; Khandelwal, U.; Levy, O.; and Manning, C. D. 2019. What does bert look at? an analysis of bert's attention. *arXiv:1906.04341*.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the NIPS*, 3123–3131.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the NAACL*, 4171–4186.

Dong, Z.; Yao, Z.; Gholami, A.; Mahoney, M.; and Keutzer, K. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. *arXiv:1905.03696*.

Gholami, A.; Kwon, K.; Wu, B.; Tai, Z.; Yue, X.; Jin, P.; Zhao, S.; and Keutzer, K. 2018. Squeezenext: Hardware-aware neural network design. In *Proceedings of the CVPR Workshops*, 1638–1647.

Hassibi, B.; Stork, D. G.; and Wolff, G. 1994. Optimal brain surgeon: Extensions and performance comparisons. In *Proceedings of the NIPS*, 263–270.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv:1503.02531*.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18(1):6869–6898.

Iandola, F. N.; Han, S.; Moskewicz, M. W.; Ashraf, K.; Dally, W. J.; and Keutzer, K. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*.

LeCun, Y.; Denker, J. S.; and Solla, S. A. 1990. Optimal brain damage. In *Proceedings of the NIPS*, 598–605.

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

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv:1907.11692*.

Ma, X.; Zhang, P.; Zhang, S.; Duan, N.; Hou, Y.; Song, D.; and Zhou, M. 2019. A tensorized transformer for language modeling. *arXiv:1906.09777*.

Michel, P.; Levy, O.; and Neubig, G. 2019. Are sixteen heads really better than one? *arXiv:1905.10650*.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the ECCV*, 525–542. Springer.

Sun, S.; Yu, C.; Zhe, G.; and Jingjing, L. 2019. Patient knowledge distillation for bert model compression. *Proceedings of the EMNLP*.

Tang, R.; Lu, Y.; Liu, L.; Mou, L.; Vechtomova, O.; and Lin, J. 2019. Distilling task-specific knowledge from bert into simple neural networks. *arXiv:1903.12136*.

Tay, Y.; Zhang, A.; Tuan, L. A.; Rao, J.; Zhang, S.; Wang, S.; Fu, J.; and Hui, S. C. 2019. Lightweight and efficient neural natural language processing with quaternion networks. *arXiv:1906.04393*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Proceedings of the NIPS*, 5998–6008.

Wang, P.; Xie, X.; Deng, L.; Li, G.; Wang, D.; and Xie, Y. 2018. Hitnet: hybrid ternary recurrent neural network. In *Proceedings of the NIPS*, 604–614.

Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; and Han, S. 2019. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the CVPR*, 8612–8620.

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

Xu, C.; Yao, J.; Lin, Z.; Ou, W.; Cao, Y.; Wang, Z.; and Zha, H. 2018. Alternating multi-bit quantization for recurrent neural networks. *arXiv:1802.00150*.

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

Yao, Z.; Gholami, A.; Lei, Q.; Keutzer, K.; and Mahoney, M. W. 2018. Hessian-based analysis of large batch training and robustness to adversaries. *arXiv:1802.08241*.

Zhang, D.; Yang, J.; Ye, D.; and Hua, G. 2018. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the ECCV*, 365–382.

Zhang, S.; Choromanska, A. E.; and LeCun, Y. 2015. Deep learning with elastic averaging sgd. In *Proceedings of the NIPS*, 685–693.

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

Zhou, A.; Yao, A.; Guo, Y.; Xu, L.; and Chen, Y. 2017. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv:1702.03044*.

Zhou, Y.; Moosavi-Dezfooli, S.-M.; Cheung, N.-M.; and Frossard, P. 2018. Adaptive quantization for deep neural network. In *Proceedings of the AAAI*.