Value-aware Quantization for Training and Inference of Neural Networks

Eunhyeok Park[1⋆], Sungjoo Yoo[1⋆], and Peter Vajda[2]

[1] Department of Computer Science and Engineering
Seoul National University
{eunhyeok.park,sungjoo.yoo}@gmail.com
[2] Mobile Vision Team, AI Camera
Facebook
vajdap@fb.com

Abstract. We propose a novel value-aware quantization which applies aggressively reduced precision to the majority of data while separately handling a small amount of large values in high precision, which reduces total quantization errors under very low precision. We present new techniques to apply the proposed quantization to training and inference. The experiments show that our method with 3-bit activations (with 2% of large ones) can give the same training accuracy as full-precision one while offering significant (41.6% and 53.7%) reductions in the memory cost of activations in ResNet-152 and Inception-v3 compared with the state-of-the-art method. Our experiments also show that deep networks such as Inception-v3, ResNet-101 and DenseNet-121 can be quantized for inference with 4-bit weights and activations (with 1% 16-bit data) within 1% top-1 accuracy drop.

Keywords: Reduced precision, quantization, training, inference, activation, weight, accuracy, memory cost, and runtime

1 Introduction

As neural networks are being widely applied to the server and edge computing, both training and inference need to become more and more efficient regarding runtime, energy consumption and memory cost. On both servers and edge devices, it is critical to reducing computation cost in order to enable fast training, e.g., on-line training of neural networks for click prediction [6, 7], and fast inference, e.g., click prediction at less than 10ms latency constraints [13] and real-time video processing at 30 frames per second [12]. Reducing computation cost is also beneficial to reducing energy consumption in those systems since the energy consumption of GPU is mostly proportional to runtime [8].

Especially, training is constrained by the memory capacity of GPU. The large batch of a deep and wide model requires large memory during training. For instance, training of a neural network for vision tasks on high-end smartphones or self-driving cars having

⋆ Most of this work was done during the authors' visit to Facebook.

4K images requires 24MB only for the input to the first layer of the network. During training, we need to store the activations of all the intermediate layers, and the required memory size (batch size x total activation size of the network) can easily exceed the GPU memory capacity. Likewise, the limited GPU capacity also restricts the large-scale training on GPU clusters [15] and the federated learning on embedded devices.

Reduced precision has potential to resolve the problems of runtime, energy consumption, and memory cost by reducing the data size thereby enabling more parallel and energy-efficient computation, e.g., four int8 operations instead of a single fp32 operation, at a smaller memory footprint. The state-of-the-art techniques of quantization are 16-bit training [4] and 8-bit inference [18]. Considering the trend of ever-increasing demand for training and inference on both servers and edge devices, further optimizations in quantization, e.g., 4 bits, will be more and more required [23].

In this paper, we propose a novel quantization method based on the fact that the distributions of weights and activations have the majority of data concentrated in narrow regions while having a small number of large values scattered in large regions. By exploiting the fact, we apply reduced precision only to the narrow regions thereby reducing quantization errors for the majority of data while separately handling large values in high precision. For very deep networks such as ResNet-152 and DenseNet-201, our proposed quantization method enables training with 3-bit activations (2% large values). Our method also offers low-precision inference with 4 to 5-bit weights and activations (1% large values) even for optimized networks such as SqueezeNet-1.1 and MobileNet-v2 as well as deeper networks.

2 Related Work

Recently, there have been presented several methods of memory-efficient training. In [2], Chen et al. propose a checkpointing method of storing intermediate activations of some layers to reduce memory cost of storing activations and re-calculating the other activations during back-propagation. In [5], Gomez et al. present a reversible network which, during back-propagation, re-computes input activations utilizing output activations thereby minimizing the storage of intermediate activations. The existing methods of checkpointing and reversible network are effective in reducing memory cost. However, they have an common critical limitation, the additional computation to re-compute activations during back-propagation. Considering that computation cost determines runtime and energy consumption of training on GPUs, the additional computation cost needs to be minimized. As will be explained in the experiments, our proposed quantization method gives much smaller cost in both memory and computation than the state-of-the-art ones. More importantly, it has a potential of offering less computation cost than the conventional training method. [3]

The state-of-the-art quantization methods of training and inference for deep networks, e.g., ResNet-152 are 16-bit training [4] and 8-bit inference [18]. In [4], Ginsburg

[3] Since the writing of this paper, Jain et al. proposed Gist system aiming to reduce activation memory footprint during training [11]. Compared to their work, we give the detailed explanation about the quantized back-propagation, and our method gives smaller memory footprint due to the benefit of value-aware quantization.

et al. propose 16-bit training based on loss scaling (for small activations or local gradients) and fp32 accumulation. In [18], Migacz proposes utilizing Kullback-Leibler (KL) divergence in determining the linear range to apply 8-bit quantization with clipping. There are studies towards more aggressive quantization for training, e.g., [3, 30]. In [3], De Sa et al. propose bit centering to exploit the fact that gradients tend to get smaller as training continues. However, these aggressive methods are limited to small networks and do not preserve full-precision accuracy for very deep models such as ResNet-152.

We classify quantization methods for inference into two types, linear and non-linear ones. The linear methods utilize uniform spacing between quantization levels, thereby being more hardware friendly, while the non-linear ones have non-uniform spacing mostly based on clustering. As the purest form of linear quantization, in [27], Rastegari et al. show that a weight binarization of AlexNet does not lose accuracy. In [9], Hubara et al. propose a multi-bit linear quantization method to offer a trade-off between computation cost and accuracy. In [30], Zhou et al. propose a multi-bit quantization which truncates activations to reduce quantization errors for the majority of data.

In [21], Miyashita et al. propose logarithm-based quantization and show that AlexNet can be quantized with 4-bit weights and 5-bit activations at 1.7% additional loss of top-5 accuracy. In [32], Zhu et al. show that deep models can be quantized with separately scaled ternary weights while utilizing full-precision activations. In [22], Park et al. propose a clustering method based on weighted entropy and show 5-bit weight and 6-bit activation can be applied to deep models such as ResNet-101 at less than 1% additional loss of top-5 accuracy. In [31], Zhou et al. propose a clustering method called balanced quantization which tries to balance the frequency of each cluster thereby improving the utility of quantization levels. Recently, several studies report that increasing the number of channels [20] and adopting teacher-student models [19, 25] help to reduce the accuracy loss due to quantization. These methods can be utilized together with our proposed quantization method.

Compared with the existing quantization methods, our proposed method, which is a linear method, enables smaller bitwidth, effectively 4 bits for inference in very deep networks such as ResNet-101 and DenseNet-121 for which there is no report of accurate 4-bit quantization in the existing works.

3 Motivation

Figure 1 (a) and (b) illustrate the distributions (y-axis in log scale) of activations and weights in the second convolutional layer of GoogLeNet. As the figures show, both distributions are wide due to a small number of large values. Given a bitwidth for low precision, e.g., 3 bits, the wider the distribution is, the larger quantization errors we obtain. Figure 1 (c) exemplifies the conventional 3-bit linear quantization applied to the distribution of activations in Figure 1 (a). As the figure shows, the spacing between quantization levels (vertical bars) is large due to the wide distribution, which incurs large quantization errors.

When comparing Figure 1 (a) and (c), it is clear that the majority of quantization levels is not fully utilized. Especially, the levels assigned to large values have much fewer data than those assigned to small values, which motivates our idea. Figure 1 (d)

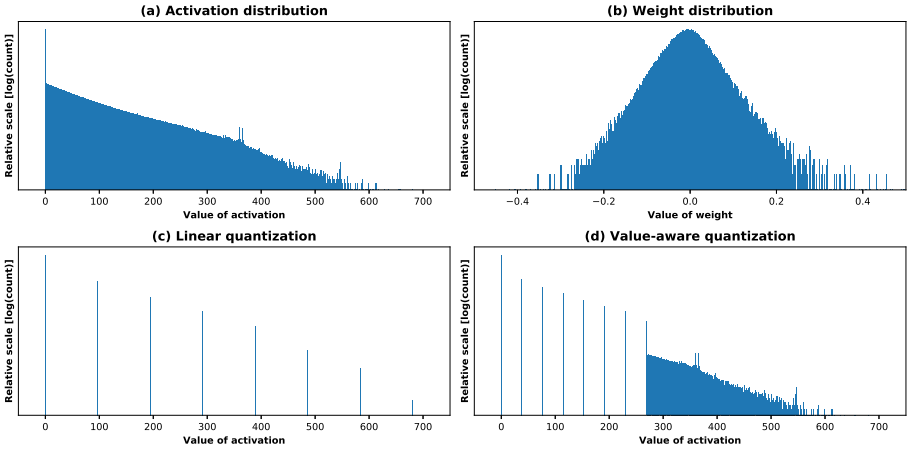

Fig. 1. Activation and weight distributions of second convolutional layer in GoogLeNet.

illustrates our idea. We propose applying low precision only to small values, i.e., the majority of data, not all. As the figure shows, the spacing between quantization levels gets much smaller than that in the conventional linear quantization in Figure 1 (c). Such a small spacing can significantly reduce the quantization error for the majority of data. Large values have the more significant impact on the quality of network output. Thus, we propose handling the remaining large values in high precision, e.g., in 32 or 16 bits. The computation and memory overhead of handling high-precision data is small because their frequency, which is called the ratio of large activations, in short, *activation ratio* (AR), is small, e.g., 1-3% of total activation data.[4]

4 Proposed Method

Our basic approach is first to perform value profiling to identify large values during training and inference. Then, we apply reduced precision to the majority of data, i.e., small ones while keeping high precision for the large values. We call this method value-aware quantization (V-Quant).

We apply V-Quant to training to reduce the memory cost of activations. We also apply it to inference to reduce the bitwidth of weights and activations of the trained neural network. To do that, we address new problems as follows.

– (Sections 4.1 and 4.2) To prevent the quality degradation of training results due to quantization, we propose a novel scheme called *quantized activation back-propagation*, in short, quantized back-propagation. We apply our quantization only to the activations used in the backward pass of training and perform forward pass with full-precision activations.

[4] We use two ratios of large values, one for large weights and the other for large activations. We use AR to denote the ratio of large activations.

– (Sections 4.4 and 4.7) Identifying large values requires sorting which is expensive. To avoid the overhead of global communication between GPUs for sorting during training, we propose performing sorting and identifying large values locally on each GPU.
– (Sections 4.5 and 4.6) We present new methods for further reduction in memory cost of training. To reduce the overhead of mask information required for ReLU function during back-propagation, we propose ReLU and value-aware quantization. For further reduction in memory cost, we also propose exploiting the fact that, as training continues, the less amount of large activations is required.

4.1 Quantized Back-Propagation

Figure 2 shows how to integrate the proposed method with the existing training pipeline. As the figure shows, we add a new component of value-aware quantization to the existing training flow. In the figure, thick arrows represent the flow of full-precision activations (in black) and gradients (in red).

First, we perform the forward pass with full-precision activations and weights, which gives the same loss as that of the existing full-precision forward pass (step 1 in the figure). During the forward pass, after obtaining the output activations of each layer, e.g., layer l, the next layer (layer $l+1$) of network takes as input the full-precision activations. Then, we apply our quantization method, RV-Quant to them (those of layer l) in order to reduce their size (step 2). As the result of the forward pass, we obtain the loss and the quantized activations.

During the backward pass, when the activations of a layer are required for weight update, we convert the quantized, mostly low-precision, activations, which are stored in the forward pass, into full-precision ones (step 3). Note that this step only converts the data type from low to high precision, e.g., from 3 to 32 bits. Then, we perform the weight update with back-propagated error (thick red arrow) and the activations (step 4).

Note that there is no modification in the computation of the existing forward and backward passes. Especially, as will be explained in the next subsection, when ReLU is used as activation function, the backward error propagation (step 5 in the figure) keeps full-precision accuracy. The added component of value-aware quantization performs conversions between full-precision and reduced-precision activations and compresses a small number of remaining large high-precision activations, which are sparse, utilizing a conventional sparse data representation, e.g., compressed sparse row (CSR).

The conversion from full to reduced precision (step 2) reduces memory cost while that from reduced to full precision (step 3) changes data type back to full precision one thereby increasing memory cost back to that of full precision. Note that the full-precision activations, obtained from the quantized ones, are discarded after weight update for their associated layer. Thus, we need memory resource for the stored quantized activations of the entire network and the full-precision input/output activations of only one layer, which we call working activations, for the forward/backward computation.

As will be explained later in this section, for further reduction in memory cost, the ReLU function consults the value-aware quantization component for the mask information which is required to determine to which neuron to back-propagate the error (step 6).

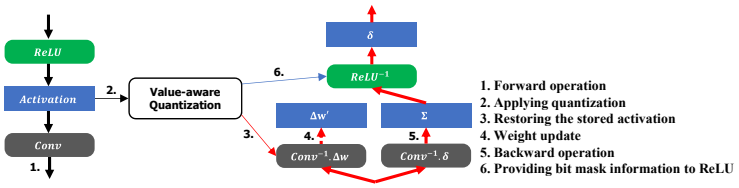

Fig. 2. Value-aware quantization in training pipeline.

4.2 Back-Propagation of Full-Precision Loss

Our proposed method can suffer from quantization error in weight update since we utilize quantized activations. We try to reduce the quantization error by applying reduced precision only to narrow regions, determined by AR, having the majority of data while separately handling the large values in high precision.

Moreover, in state-of-the-art networks where ReLU is utilized as activation function, the back-propagated error is not affected by our quantization of activations as is explained below. Equation (1) shows how we calculate weight update during back-propagation for a multilayer perceptron (MLP).

$$\Delta w_{ji} = \eta \delta_j y_i \tag{1}$$

where Δw_{ji} represents the update of weight from neuron i (of layer l) to neuron j (of layer $l+1$), η learning rate, δ_j the local gradient of neuron j (back-propagated error to this neuron), and y_i the activation of neuron i. Equation (1) shows that the quantization error of activation y_i can affect the weight update. In order to reduce the quantization error in Equation (1), we apply V-Quant to activations y_i.

The local gradient δ_j is calculated as follows.

$$\delta_j = \varphi'(v_j) \cdot (\Sigma_k \delta_k w_{kj}) \tag{2}$$

where $\varphi'()$ represents the derivative of activation function, v_j the input to neuron j and w_{kj} the weight between neuron j (of layer $l+1$) to neuron k (of layer $l+2$). Equation (2) shows that the local gradient is a function of the input to neuron, v_j which is the weighted sum of activations. However, if ReLU is used as the activation function, then $\varphi'()$ becomes 1 yielding $\delta_j = \varphi'(v_j) \cdot (\Sigma_k \delta_k w_{kj}) = \Sigma_k \delta_k w_{kj}$, which means *the local gradient becomes independent of activations*. Thus, aggressive quantizations of intermediate activations, e.g., 3-bit activations can hurt only the weight update in Equation (1), not the local gradient in Equation (2). This is the main reason why our proposed method can offer full-precision training accuracy even under aggressive quantization of intermediate activations as will be shown in the experiments.

4.3 Potential of Further Reduction in Computation Cost

Compared with the existing methods of low memory cost in training [2] [5], our proposed method reduces computation cost by avoiding re-computation during back-propagation. More importantly, our proposed method has a potential for further reduction in computation cost especially in Equation (1), though we have not yet realized

the speedup potential in this paper. It is because the activation y_i is mostly in low precision in our method. Thus, utilizing the capability of 8-bit multiplication on GPUs, our method can transform a single 16-bit x 16-bit multiplication in Equation (1) into an 8-bit x 16-bit multiplication. In state-of-the-art GPUs, we can perform two 8-bit x 16-bit multiplications at the same computation cost, i.e., execution cycle, of one 16-bit x 16-bit multiplication, which means our proposed method can double the performance of Equation (1) on the existing GPUs.

Assuming that the forward pass takes M multiplications, the backward pass takes $2M$ multiplications while each of Equations (1) and (2) taking M multiplications, respectively. Thus, the 2x improvement in computation cost of Equation (1) can reduce by up to 1/6 total computation cost of training. In order to realize the potential, further study is needed to prove that our proposed method enables 8-bit low-precision activations (with a small number of 16-bit high-precision activations) without losing the accuracy of 16-bit training [4].

Although our method can currently reduce computation cost utilizing only 8-bit multiplications on GPUs, its reduced-precision computation, e.g., 3-bit multiplications, offers opportunities of further reduction in computation cost for training in future hardware platforms supporting aggressively low precision, e.g., [28].

4.4 Local Sorting in Data Parallel Training

V-Quant requires sorting activations. Assuming that we adopt data parallelism in multi-GPU training, the sorting can incur significant overhead in training runtime since it requires exchanging the activations of each layer between GPUs. What is worse, in reality, such a communication is not easily supported in some training environments, e.g., PyTorch, and it restrict the scalability in any training environments including TensorFlow , Caffe2 and others as well as PyTorch. In order to address the problem of activation exchange, we propose performing sorting locally on each GPU, which eliminates inter-GPU communication for activation exchange. Then, each GPU performs V-Quant locally by applying the same AR, i.e., the same ratio of large activations. Compared with the global solution that collects all the activations and applies the AR to the global distribution of activations, the proposed local solution can lose accuracy in selecting large values. However, our experiments show that the proposed method of local sorting works well, which means that the selection of large values does not need to be accurate.

4.5 ReLU and Value-aware Quantization (RV-Quant)

The error is back-propagated through the neurons the output activations of which are non-zero. The zero activations result from quantization (called *quantization-induced zero*) as well as ReLU activation function. In order to realize the same back-propagation as the full-precision training, it is required to back-propagate errors in the case of quantization-induced zero. To identify the neurons having quantization-induced zero, we would need a bit mask, i.e., 1-bit memory cost for a neuron. In case that the activations are quantized at a minimal number of bits, e.g., 3 bits, the overhead of the mask bit is significant, e.g., one additional bit for 3-bit activation on each neuron. To reduce the overhead of the mask bit, we exploit the fact that the mask bit is needed only for the

case of zero activation. We allocate two states to represent two different types of zero values, i.e., original zero and quantization-induced zero. Thus, given K bits for low precision, we allocate two of 2^K quantization levels to the zero values while representing the positive activation values with $2^K - 2$ levels. We call this quantization ReLU and value-aware quantization (RV-Quant). As will be shown in the experiments, RV-Quant removes the overhead of mask bit while keeping training accuracy.

4.6 Activation Annealing

According to our investigation, the required amount of large activations varies across training phases. To be specific, the early stage of training tends to require more large activations while the later stage tends to need less large activations. We propose exploiting the fact and adjusting AR in a gradual manner from large to small AR across training phases, which we call *activation annealing*. As will be shown in the experiments, activation annealing can maintain training quality while reducing the average memory cost across the entire training phases.

4.7 Quantized Inference

In order to obtain quantized neural networks for inference, we perform V-Quant as a post-processing of training, i.e., we apply V-Quant to the weights and activations of trained networks. To recover from the accuracy loss due to quantization, we perform fine-tuning as follows. We perform forward pass while utilizing the quantized network, i.e., applying V-Quant to weights and activations. During back-propagation, we update full-precision weights. As will be shown in the experiments, the fine-tuning incurs a minimal overhead in training time, i.e., only a few additional epochs of training. Note that we apply local sorting in Section 4.4 to avoid communication overhead when multiple GPUs are utilized in fine-tuning.

During fine-tuning, we evaluate candidate ratios for large weights and activations and, among those candidates, select the best configuration which minimizes the bitwidth while meeting accuracy requirements. Note that, as will be explained in the experiments, the total number of candidate combinations is small.

In order to identify large activations meeting the AR, we need to sort activations, which can be expensive in inference. In order to avoid the sorting overhead, we need low-cost sorting solutions, e.g., sampling activations to obtain an approximate distribution of activations. Detailed implementations of quantized models including the low-cost sorting are beyond the scope of this paper and left for further study.

5 Experiments

We evaluate our proposed method on ImageNet classification networks, AlexNet, VGG-16, SqueezeNet-1.1, MobileNet-v2, Inception-v3, ResNet-18/50/101/152 and DenseNet-121/201. We test the trained/quantized networks with ILSVRC2012 validation set (50k images) utilizing a single center crop of 256x256 resized image. We

also use an LSTM for word-level language modeling [10, 26, 29]. We implemented our method on PyTorch framework [24] and use the training data at Torchvision [16].

The initial learning rate is set to 0.1 (ResNet-18/50/152 and DenseNet-201), or 0.01 (AlexNet and VGG-16). The learning rate is decreased by 10x at every multiple of 30 epochs and the training stops at 90 epochs. In SqueezeNet-1.1, MobileNet-v2, and Inception-v3, we use the same parameters in the papers except that we use a mini-batch of 256 and SGD instead of RMSprop. In addition, we replace ReLU6 in MobileNet-v2 with ReLU to apply V-Quant.

We apply V-Quant and RV-Quant to training to minimize memory cost. During training, in order to compress the sparse large activations on GPU, we use the existing work in [1]. To obtain quantized networks for inference, we perform fine-tuning with V-Quant for a small number of additional epochs, e.g., 1-3 epochs after total 90 epochs of original training. All networks are initialized in the same condition.

We compare classification accuracy between full-precision models and those under RV-Quant (training) and V-Quant (training/inference). For each network, we use the same randomly initialized condition and perform training for different RV-Quant and V-Quant configurations.

5.1 Training Results

Table 1 shows top-1/top-5 accuracy of ResNet-50 obtained, under V-Quant, varying the bitwidth of low-precision activation and the ratio of large activation, AR. The table shows that the configuration of 3-bit activations with the AR of 2% (in bold) gives training results equivalent to the full-precision (32-bit) training in terms of top-1 accuracy, which corresponds to 6.1X (=1/((3+1)/32 + 0.04)) reduction in the memory cost of stored activation at the same quality of training.[5] The table also shows that a very aggressive quantization of 2-bit activation and 1% AR loses only 0.27%/0.24% in top-1/top-5 accuracy, which is comparable to the case of 5-bit quantization without large values (5-bit with AR 0% in the table).

Note that the total memory cost of activations includes that of stored activations of the entire network and that of full-precision working activations (input to the associated layer) required for weight update. Thus, the above-mentioned reduction of 6.1X is only for the memory cost of stored activations. We will give the comparison of the total memory cost of activations later in this section. In addition, we do not compare the accuracy of the state-of-the-art method [2] since it provides the same accuracy as full-precision training. However, we provide the memory consumption of this method later in this section, including that of the conventional linear quantization method (8-bit with AR 0%).

Table 2 shows top-1/top-5 accuracy of ResNet-50 under RV-Quant. As the table shows, RV-Quant gives similar results to V-Quant, e.g., top-1 accuracy of 3-bit 2% RV-Quant gives an equivalent result to full precision. Compared with V-Quant, RV-Quant

[5] Note that V-Quant still requires 1-bit mask information for each neuron. In addition, the sparse data representation of large values, e.g., CSR doubles the size of the original sparse data yielding the memory cost of 4% with the AR of 2%.

AR [%]	0	1	2	3	4	5
1-bit	5.30 / 15.23	74.51 / 92.05	75.17 / 92.50	75.21 / 92.48	75.70 / 92.66	75.57 / 92.66
2-bit	65.75 / 86.72	75.65 / 92.66	75.64 / 92.70	75.66 / 92.51	75.34 / 92.66	75.58 / 92.62
3-bit	75.49 / 92.61	75.71 / 92.59	**75.92 / 92.86**	75.93 / 92.96	75.89 / 92.94	75.73 / 92.63
4-bit	75.70 / 92.75	75.78 / 92.67	75.88 / 92.93	75.79 / 92.71	75.85 / 92.69	75.92 / 92.86
5-bit with AR 0 %	75.60 / 92.61		**6-bit with AR 0 %**	75.92 / 92.83		
7-bit with AR 0 %	75.89 / 92.79		**8-bit with AR 0 %**	75.67 / 92.85		

Table 1. Top-1/top-5 accuracy [%] of ResNet-50 with various bitwidth & AR configurations. The full-precision network gives the accuracy of 75.92/92.90%.

reduces the memory cost by 1 bit per neuron. Thus, the configuration of 3-bit 2% RV-Quant gives 7.5X (=1/(3/32 + 0.04)) reduction in the memory cost of stored activations. In addition, we can further reduce the memory cost of stored activations by applying traditional compression techniques to the reduced-precision activations. In the case of 3-bit 2% RV-Quant for ResNet-50, by applying Lempel-Ziv compression, we can further reduce the memory cost of the 3-bit data by 24.4%, which corresponds to 9.0x reduction in the memory cost of the whole stored activations.

AR [%]	0	1	2	3	4	5
2-bit	35.52 / 60.86	75.34 / 92.56	75.41 / 92.49	75.67 / 92.59	75.50 / 92.46	75.27 / 92.65
3-bit	75.16 / 92.55	75.88 / 92.80	75.93 / 92.70	75.66 / 92.74	75.91 / 92.75	75.49 / 92.58

Table 2. Top-1/top-5 accuracy [%] of ResNet-50 under RV-Quant. The full-precision network gives the accuracy of 75.92/92.90%.

Table 3 compares the accuracy of neural networks under full-precision training and two RV-Quant configurations. As the table shows, 3-bit 2% RV-Quant gives almost the same training accuracy as full-precision training for all the networks.

	AlexNet	ResNet-18	SqueezeNet-1.1	MobileNet-v2	VGG-16	Inception-v3	ResNet-152	DenseNet-201
Full	56.35 / 79.02	69.91 / 89.38	58.67 / 81.05	70.10 / 89.74	71.86 / 90.48	74.19 / 91.92	77.95 / 94.02	77.42 / 93.59
3-bit 2%	56.14 / 78.99	69.92 / 89.23	58.53 / 80.94	70.12 / 89.76	71.74 / 90.46	74.14 / 91.92	77.76 / 93.89	77.28 / 93.44
8-bit 0%	56.24 / 78.95	70.01 / 89.28	58.75 / 81.29	70.29 / 89.64	71.77 / 90.66	74.22 / 92.08	78.35 / 93.95	77.32 / 93.51

Table 3. Training results. Full means the results of conventional full-precision training, while 3-bit 2% and 8-bit 0% correspond to RV-Quant. The full-precision network gives the accuracy of 75.92/92.90%.

Table 4 compares the total memory cost of activations (both stored quantized and full-precision working activations) in training with 256 mini-batch sizes. We compare two existing methods and three RV-Quant configurations. 'Full' represents the memory cost of conventional training with full-precision activation. As a baseline, we use the checkpointing method of Chen et al. [2] since it is superior to others including [5],

especially for deep neural networks. We calculate the memory cost of the checkpointing method to account for the minimum amount of intermediate activations to re-compute correct activations while having the memory cost of O($\sqrt{N}$) where N is the number of layers [2].

The table shows that, compared with the checkpointing method, RV-Quant gives significant reductions in the total memory cost of activations. For instance, in the case of ResNet-152 which is favorable to the checkpointing method due to the simple structure as well as a large number of layers, ours reduces the memory cost by 41.6% (from 5.29GB to 3.09GB). In networks having more complex sub-networks, e.g., Inception modules, ours gives more reductions. In the case of Inception-v3, ours gives a reduction of 53.7% (3.87GB to 1.79GB). Note that in the case of AlexNet, the reduction is not significant. It is because the input data occupy the majority of stored activations and we store them in full precision. However, the impact of input data storage diminishes in deep networks.

We also measured the training runtime of ResNet-50 with mini-batch of 64 on NVIDIA Tesla M40 GPU. Compared to the runtime of existing full-precision training, our method requires a small additional runtime, 8.8% while the checkpointing method has much larger runtime overhead, 32.4%. Note that as mentioned in Section 4.3, our method has a potential for further reduction in training time on hardware platforms supporting reduced-precision computation.

	AlexNet	ResNet-18	SqueezeNet-1.1	MobileNet-v2	ResNet-50	VGG-16	Inception-v3	ResNet-152	DenseNet-201
Full	0.35	1.86	1.58	7.34	9.27	9.30	9.75	20.99	24.53
Chen et al. [2]	x	0.98 (52.1 %)	1.05 (66.9 %)	4.21 (52.1 %)	3.70 (39.9 %)	x	3.87 (39.8 %)	5.29 (25.2 %)	6.62 (27.0 %)
(2,0)	0.23 (66.4 %)	0.42 (22.6 %)	0.59 (37.5 %)	0.74 (10.0 %)	1.22 (13.2 %)	3.65 (39.2 %)	1.16 (11.9 %)	1.64 (7.78 %)	2.09 (8.51 %)
(3,0)	0.23 (67.8 %)	0.46 (24.3 %)	0.61 (38.8 %)	0.84 (11.4 %)	1.34 (14.5 %)	3.75 (40.3 %)	1.43 (14.8 %)	2.27 (10.8 %)	2.85 (11.6 %)
(3,2)	0.24 (69.5 %)	0.50 (26.5 %)	0.64 (40.4 %)	1.13 (15.4 %)	1.52 (16.4 %)	3.88 (41.7 %)	1.79 (18.4 %)	3.09 (14.7 %)	3.83 (15.6 %)

Table 4. Comparison of memory cost (in GB).

Table 5 shows the impact of RV-Quant configurations on training accuracy of ResNet-50. We change the configurations when the learning rate changes (with the initial value of 0.1) at 0.01 and 0.001. For instance, (F)-(3,2)-(2,0) represents the case that, as the initial configuration, we use full-precision activation (F) during back-propagation. After 30 epochs, the configuration is changed to 3-bit 2% RV-Quant. Then, after 60 epochs, it is changed to 2-bit 0% RV-Quant.

In Table 5, the key observation is that it is important to have high precision at the beginning of training. Compared with the case that training starts with full-precision activations and ends with aggressively reduced precision, (F)-(3,2)-(2,0), the opposite case, (2,0)-(3,2)-(F) gives significantly lower accuracy, 75.45% vs. 50.36%. Another important observation is that activation annealing works. For instance, (3,2)-(3,1)-(3,0) gives almost the same result to (3,2)-(3,2)-(3,2) in Table 2 and, a more aggressive case, (3,2)-

Configuration	Accuracy	Configuration	Accuracy	Configuration	Accuracy	Configuration	Accuracy
(3,2)-(2,1)-(2,0)	75.01 / 92.42	**(2,0)-(2,1)-(3,2)**	47.35 / 72.31	**(3,2)-(3,1)-(3,0)**	75.72 / 92.69	**(3,0)-(3,1)-(3,2)**	75.60 / 92.77
(F)-(3,2)-(2,0)	75.45 / 92.63	**(2,0)-(3,2)-(F)**	50.36 / 75.02	**(3,2)-(3,1)-(2,0)**	75.34 / 92.55	**(2,0)-(3,1)-(3,2)**	48.67 / 73.54
(F)-(2,1)-(2,0)	75.38 / 92.44	**(2,0)-(2,1)-(F)**	52.72 / 76.76				

Table 5. Sensitivity analysis of RV-Quant configurations (bitwidth and AR [%]) across training phases. The full-precision network gives the accuracy of 75.92/92.90%.

(3,1)-(2,0) gives only by 0.58% smaller top-1 accuracy. Thus, as training advances, we need the smaller amount of large values, which means we can have smaller memory cost of activations. This can be exploited for memory management in servers. We expect it can also be utilized in memory-efficient server-mobile co-training in federated learning [14] where the later stage of training requiring smaller memory cost can be performed on memory-limited mobile devices while meeting the requirements of user-specific adaptation using private data.

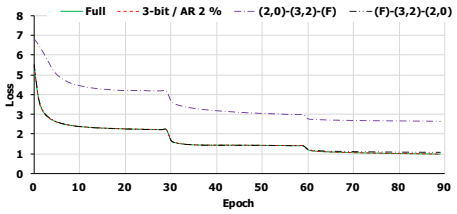

Fig. 3. Training loss of ResNet-50 with various RV-Quant configurations.

Figure 3 shows the training loss of different RV-Quant configurations during training. First, the figure shows that too aggressive quantization at the beginning of training, i.e., (2,0)-(3,2)-(F), does not catch up with the loss of full-precision training (Full in the figure). The figure also shows that the configuration of 3-bit 2% RV-Quant gives almost the same loss as the full-precision training.

5.2 Inference Results

Figure 4 shows the accuracy of quantized models across different configurations of bitwidth and AR. We apply the same bitwidth of low precision to both weights and activations and 16 bits to large values of weights and activations. In addition, we quantize all the layers including the first (quantized weights) and last convolutional layers. As the figure shows, V-Quant with fine-tuning, at 4 bits and an AR of 1%, gives accuracy comparable to full precision in all the networks within 1% of top-1 accuracy. If V-Quant is applied without fine-tuning, the larger AR needs to be used to compensate for accuracy drop due to quantization. However, the figure shows that fine-tuning successfully closes the accuracy gap between V-Quant and full-precision networks.

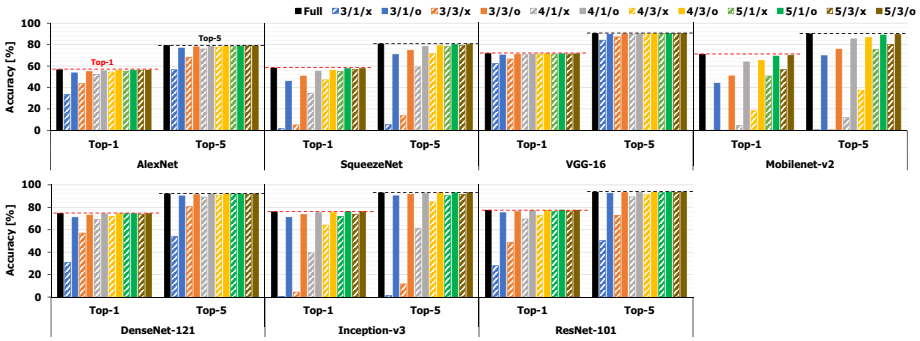

Fig. 4. V-Quant results. The dashed lines and black solid bar represent full-precision accuracy. Legend: bitwidth/AR [%]/fine-tuning or not.

Figure 5 illustrates the effect of large values on the classification ability. The figure shows the principal component analysis (PCA) results of the last convolutional layer of AlexNet for four classes. Figure 5 (a) shows the PCA result of full-precision network. As Figure 5 (b) shows, when the conventional 4-bit linear quantization, or 4-bit 0% V-Quant is applied to weights/activations, it is difficult to classify four groups of data successfully. This is because the quantization errors are accumulated across layers thereby deteriorating the quality of activations. However, as Figure 5 (c) shows, only a very small amount (0.1%) of large values can improve the situation. As more large values are utilized, the classification ability continues to improve (3% in Figure 5 (d)). The figure demonstrates that our idea of reducing quantization errors for the majority of data by separately handling large values is effective in keeping good representations.

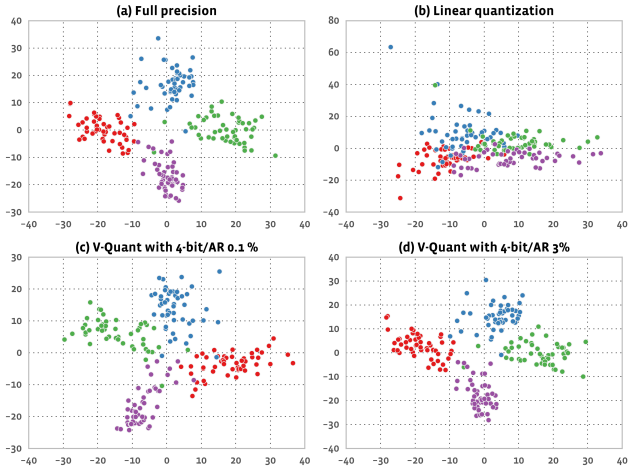

Fig. 5. PCA analysis of the input activations on the last fully-connected layer of AlexNet.

5.3 LSTM Language Model

We apply V-Quant to an LSTM for word-level language modeling [10,26,29]. Table 6 shows the results of the models. Each of the large and small models has two layers. The large model has 1,500 hidden units and the small one 200 units. We measure word-level perplexity on Penn Tree Bank data [17]. We apply V-Quant only to the weights of the models since clipping is applied to the activation.[6]

As Table 6 shows, we evaluate three cases of bitwidth, 2, 3 and 4 bits and two ratios of large weights, 1%, and 3%. As the table shows, for the large model, the 4-bit 1% V-Quant preserves the accuracy of the full-precision model. However, the small model requires the larger ratio of large weights (3%) in order to keep the accuracy.

	Large-1%		**Large-3%**		**Small-1%**		**Small-3%**	
	Valid	**Test**	**Valid**	**Test**	**Valid**	**Test**	**Valid**	**Test**
float	75.34	72.31	75.34	72.31	103.64	99.24	103.64	99.24
2-bit	79.92	77.31	77.87	74.99	140.70	135.11	122.25	117.76
3-bit	76.19	73.22	75.79	72.72	107.60	102.82	105.99	101.44
4-bit	75.46	72.48	75.44	72.44	104.22	99.83	103.95	99.57

Table 6. Impact of quantization on word-level perplexity of an LSTM for language modeling.

6 Conclusions

We presented a novel value-aware quantization to reduce memory cost in training and computation/memory cost in inference. To realize aggressively low precision, we proposed separately handling a small number of large values and applying reduced precision to the majority of small values, which contributes to reducing quantization errors. In order to apply our idea to training, we proposed quantized back-propagation which utilizes quantized activations only during back-propagation. For inference, we proposed applying fine-tuning to quantized networks to recover from accuracy loss due to quantization. Our experiments show that our proposed method outperforms the state-of-the-art method of low-cost memory in training in deep networks, e.g., 41.6% and 53.7% smaller memory cost in ResNet-152 and Inception-v3, respectively. It also enables 4-bit inference (with 1% large values) for deep networks such as ResNet-101 and DenseNet-121, and 5-bit inference for efficient networks such as SqueezeNet-1.1 and MobileNet-v2 within 1% of additional top-1 accuracy loss.

Acknowledgment

This work was supported in part by National Research Foundation of Korea (NRF-2016M3A7B4909604).

[6] The distribution of activations obtained by clipping tends to have the large population near the maximum/minimum values. Considering that clipped activation functions like ReLU6 are useful, it will be interesting to further investigate clipping-aware quantization.

References

1. Bakunas-Milanowski, D., et al.: Efficient algorithms for stream compaction on gpus. International Journal of Networking and Computing (IJNC) (2017)
2. Chen, T., et al.: Training deep nets with sublinear memory cost. arXiv:1604.06174 (2016)
3. De Sa, C., et al.: High-accuracy low-precision training. arXiv:1803.03383 (2018)
4. Ginsburg, B., et al.: NVIDIA Mixed Precision Training on Volta GPUs. GPU Technology Conference (2017)
5. Gomez, A.N., et al.: The reversible residual network: Backpropagation without storing activations. Advances in Neural Information Processing Systems (NIPS) (2017)
6. Hazelwood, K., et al.: Applied machine learning at facebook: A datacenter infrastructure perspective. International Symposium on High-Performance Computer Architecture (HPCA) (2018)
7. He, X., et al.: Practical lessons from predicting clicks on ads at facebook. International Workshop on Data Mining for Online Advertising (ADKDD) (2014)
8. Hong, S., Kim, H.: An integrated GPU power and performance model. International Symposium on Computer Architecture (ISCA) (2010)
9. Hubara, I., et al.: Quantized neural networks: Training neural networks with low precision weights and activations. arXiv:1609.07061 (2016)
10. Inan, H., Khosravi, K., Socher, R.: Tying word vectors and word classifiers: A loss framework for language modeling. arXiv:1611.01462 (2016)
11. Jain, A., et al.: Gist: Efficient data encoding for deep neural network training. International Symposium on Computer Architecture (ISCA) (2018)
12. Jia, Y., Peter, V.: Delivering real-time ai in the palm of your hand. `https://code.facebook.com/posts/196146247499076/delivering-real-time-ai-in-the-palm-of-your-hand/`, accessed: 2018-7-25
13. Jouppi, N.P., et al.: In-datacenter performance analysis of a tensor processing unit. International Symposium on Computer Architecture (ISCA) (2017)
14. Konečnỳ, J., et al.: Federated learning: Strategies for improving communication efficiency. arXiv:1610.05492 (2016)
15. Mahajan, D., et al.: Exploring the limits of weakly supervised pretraining. arXiv:1805.00932 (2018)
16. Marcel, S., Rodriguez, Y.: Torchvision the machine-vision package of torch. ACM Multimedia (2010)
17. Marcus, M.P., Marcinkiewicz, M.A., Santorini, B.: Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics (1993)
18. Migacz, S.: NVIDIA 8-bit inference width TensorRT. GPU Technology Conference (2017)
19. Mishra, A., Marr, D.: Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. arXiv:1711.05852 (2017)
20. Mishra, A., et al.: Wrpn: Wide reduced-precision networks. arXiv:1709.01134 (2017)
21. Miyashita, D., Lee, E.H., Murmann, B.: Convolutional neural networks using logarithmic data representation. arXiv:1603.01025 (2016)
22. Park, E., Ahn, J., Yoo, S.: Weighted-entropy-based quantization for deep neural networks. Computer Vision and Pattern Recognition (CVPR) (2017)
23. Park, E., Kim, D., Yoo, S.: Energy-efficient neural network accelerator based on outlier-aware low-precision computation. International Symposium on Computer Architecture (ISCA) (2018)
24. Paszke, A., et al.: Pytorch (2017)
25. Polino, A., Pascanu, R., Alistarh, D.: Model compression via distillation and quantization. International Conference on Learning Representation (ICLR) (2018)

26. Press, O., Wolf, L.: Using the output embedding to improve language models. European Chapter of the Association for Computational Linguistics (EACL) (2017)
27. Rastegari, M., et al.: Xnor-net: Imagenet classification using binary convolutional neural networks. European Conference on Computer Vision (ECCV) (2016)
28. Umuroglu, Y., et al.: Finn: A framework for fast, scalable binarized neural network inference. Architecture of Field-Programmable Gate Arrays (FPGA) (2017)
29. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. arXiv:1409.2329 (2014)
30. Zhou, S., et al.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv:1606.06160 (2016)
31. Zhou, S., et al.: Balanced quantization: An effective and efficient approach to quantized neural networks. Journal of Computer Science and Technology (2017)
32. Zhu, C., et al.: Trained ternary quantization. arXiv:1612.01064 (2016)