Adaptive Quantization of Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Despite the state-of-the-art accuracy of Deep Neural Networks (DNN) in various classification problems, their deployment onto resource constrained edge computing devices remains challenging, due to their large size and complexity. Several recent studies have reported remarkable results in reducing this complexity through quantization of DNN models. However, these studies usually do not consider the change in loss when performing quantization, nor do they take the disparate importance of DNN connections to the accuracy into account. We address these issues in this paper by proposing a new method, called adaptive quantization, which simplifies a trained DNN model by finding a unique, optimal precision for each connection weight such that the increase in loss is minimized. The optimization problem at the core of this method iteratively uses the loss function gradient to determine an error margin for each weight and assign it a precision accordingly. Since this problem uses linear functions, it is computationally cheap and, as we will show, has a closed-form approximate solution. Experiments on MNIST, CIFAR, and SVHN datasets showed that the proposed method can achieve near or better than state-of-the-art reduction in model size with similar error rate. Furthermore, it can achieve compressions close to floating-point model compression methods without loss of accuracy.

1 Introduction

Deep Neural Networks (DNNs) have achieved incredible accuracy in applications ranging from computer vision (Simonyan and Zisserman 2014) to speech recognition (Hinton et al. 2012) and natural language processing (Devlin et al. 2014). One of the key enablers of the unprecedented success of DNNs is the availability of *very large model sizes*. While the increase in model size improves the classification accuracy, it inevitably increases the computational complexity and memory requirement needed to train and store the network. This poses challenges in deploying these large models in resource-constrained edge computing environments, such as mobile devices. This challenge motivates *neural network compression*, which exploits the redundancy of neural networks to achieve drastic reductions in model sizes. The state-of-the-art neural network compression techniques include weight quantization (Courbariaux et al. 2015), weight pruning (Han et al. 2015), weight sharing (Han et al. 2015), and low rank approximation (Zhao et al. 2017). For instance, weight quantization has previously shown good accuracy with fixed-point 16-bit and 8-bit precisions (Suda et al. 2016; Qiu et al. 2016). Recent work attempts to push that even further towards reduced precision and has trained models with 4-bit, 2-bit, and 1-bit precisions using quantized training methods (Hubara et al. 2016a; Zhou et al. 2016; Hubara et al. 2016b; Courbariaux et al. 2015).

Although these quantization methods can significantly reduce model complexity, they generally have two key constraints. First, while performing quantization they do not take the change in the loss function into account. This can result in a disparity between the quantized model accuracy and the objective accuracy. Previous works get around this issue by devising quantized learning schemes that

Submitted to 6th Conference on Learning Representations (ICLR 2018). Do not distribute.

essentially try to solve the problem after the fact. Such schemes have the disadvantage of converging very slowly compared to full-precision learning methods. Second, they do not differentiate between different connections in their approaches. This is despite the fact that it has been shown that different weights are not equally important to the accuracy of the network (). Disregarding this difference constraints these methods to either use higher precisions for quantization or use deeper networks. Both of these solutions diminish the compression.

In this paper, we address the aforementioned issues by proposing *adaptive quantization*. To take the disparate importance of different connections into account, this method quantizes each weight of a trained network by a unique precision. This way, weights that impact the accuracy the most can be represented using a higher precision, while low-impact weights are represented with only fewer bits or are pruned. Consequently, it can reduce the model size significantly while maintaining a certain accuracy. The proposed method monitors the accuracy by incorporating the loss function into an optimization problem to minimize the models. The output of the optimization problem is an error margin associated to each weight. This margin is computed based on the loss function gradient of the weight and is used to determine its precision. We will show that the proposed optimization problem has a closed-form approximate solution, which can be iteratively applied to the same network to minimize its size, constraining the loss. We test the proposed method using three classification benchmarks comprising MNIST, CIFAR-10, and SVHN. We show that, across all these benchmarks, we can achieve near or better compressions compared to state-of-the-art quantization techniques. Furthermore, we can achieve similar compressions compared to state-of-the-art pruning and weight-sharing techniques which inherently require more computational resources for inference.

2 Proposed Quantization Algorithm

Despite their remarkable classification accuracies, large DNNs assimilate redundancies. Several recent works have studied these redundancies by abstracting the network from different levels and search for, in particular, redundant filters (DenseNets) and redundant connections (Deep Compression). In this work, we present an even more fine-grained study of redundancy and extend it to fixed-point quantization of network parameters. That is, we approximate the minimum size of the network when each parameter is allowed to have a distinct number of precision bits. Our goal here is to represent parameters with high precisions only when they are critical to the accuracy of the network. In this sense, our approach is similar to the weight pruning which eliminated all but the essential parameters, producing a sparse network. In the rest of this section, we first formally define the problem of minimizing the network size as an optimization problem. Then, we propose a trust region technique to approximately solve this problem. We will show that, in each iteration of the trust region method, our approach has a straightforward closed-form solution. Finally, we will explain how the hyper parameters in the algorithm are chosen and discuss the implications of the proposed technique.

2.1 Problem Definition

A formal definition of the optimization problem that was discussed previously is presented here. Then, key characteristics of the objective function are derived. We will use these characteristics later when we solve the optimization problem.

We minimize the aggregate bit-width of all network parameters while monitoring the training loss function. Due to the quantization noise, this function deviates from its optimum which was achieved by training. This effect can be controlled by introducing an upper bound on the loss function in order to maintain it reasonably close to the optimum. Consequently, solution of this minimization problem represents critical parameters with high precision to sustain high accuracy and assigns low precisions to ineffectual ones or prunes them. The problem described here can be defined in formal terms as below.

$$\min_{W} N_Q(W) = \sum_{i=1}^{n} N_q(\omega_i) \tag{1}$$

$$l(W) \leq \bar{l} \tag{2}$$

Here, n is the number of model parameters, $W = [\omega_1 ... \omega_n]^T$ is a vector of model parameters, ω_i is the i-th model parameter, and the function $N_q(\omega_i)$ is the minimum number of bits required to represent ω_i in its fixed-point format. As a result, $N_Q(W)$ is the total number of bits required to represent the model. In addition, $l(W)$ is the loss of the model over the training set (or a minibatch of the training set). $\bar{l}$ ($\geq l(W_0)$) is a constant upper bound on the loss of the neural network, which is used to bound the accuracy loss, and W_0 is the vector of initial model parameters before quantization.

The optimization problem presented in equations 1 and 2 is difficult to solve because of its non-smooth objective function. However, a smooth upper limit can be found for it.

Lemma 2.1. *Given a vector of network parameters W, a vector of tolerance values $T = [\tau_1 ... \tau_n]^T$, and a vector of quantized network parameters $W_q = [\omega_1^q ... \omega_n^q]^T$, such that each $\omega_i^q, i \in [1, n]$ solves the constrained optimization function:*

$$\min_{\omega^q} N_q(\omega^q) \tag{3}$$

$$|\omega^q - \omega_i| \leq \tau_i \tag{4}$$

We have that:

$$N_Q(W) \leq N_Q^u(T) \tag{5}$$

Where $N_Q^u(T)$ is a smooth function, defined as:

$$N_Q^u(T) = -\sum_{i=1}^{n} \log_2 \tau_i \tag{6}$$

Proof. The problem of equation 4 can be easily solved using algorithm 1, which simply checks all possible solutions one-by-one. Here, we allocate at most 32 bits to each parameter. This should be more than enough as previous works have demonstrated that networks can be easily quantized to even 8 bits without significant loss of accuracy.

Algorithm 1 Quantization of a parameter

```
procedure QUANTIZE_PARAMETER(ω_i, τ_i)
    ω^q ← 0
    N_q ← 0
    while N_q ≤ 32 do
        if |ω^q − ω_i| ≤ τ_i then
            break
        end if
        ω^q ← round(2^{N_q} ω)/2^{N_q}
        N_q ← N_q + 1
    end while
    return N_q, ω^q
end procedure
```

Algorithm 1 estimates ω_i with a rounding error of up to $\frac{1}{2^{N_q+1}}$. This error needs to be consistent with the constraint of equation 4:

$$\frac{1}{2^{N_q(\omega_i)+1}} \leq \tau_i \Rightarrow N_q(\omega_i) + 1 \geq -\log_2 \tau_i \tag{7}$$

The minimization in equation 4 entails that:

$$N_q(\omega_i) = -\lfloor \log_2 \tau_i \rfloor - 1 \leq -\log_2 \tau_i \tag{8}$$

Consequently:

$$N_Q(W) = \sum_{i=1}^{n} N_q(\omega_i) \leq -\sum_{i=1}^{n} \log_2 \tau_i = N_Q^u(T) \tag{9}$$

□

2.2 Solving the Optimization Problem

The optimization problem of equations 1 and 2 presents two key challenges. First, as was mentioned previously, the objective function is non-smooth. Second, not only is the constraint (equation 2) non-linear, but also it is unknown. In this section, we present a method that circumvents these challenges and provides an approximate solution for this problem.

We sidestep the first challenge by using the upper bound of the objective function, $N_Q^u(T)$. Particularly, we approximate the minimum of $N_Q(W)$ by first finding the optimal T for $N_Q^u(T)$ and then calculating the quantized parameters using algorithm 1. This optimization problem can be defined as: minimize $N_Q^u(T)$ such that if each ω_i is perturbed by, at most, τ_i, the loss constraint of equation 2 is not violated:

$$\min_T N_Q^u(T) \tag{10}$$

$$l(W_0 + \Delta W) \leq \bar{l} \tag{11}$$

$$\forall \Delta W = [\Delta\omega_1 ... \Delta\omega_n]^T \quad \textit{such that} \quad \forall i \in [1, n] : |\Delta\omega_i| \leq \tau_i \tag{12}$$

It is important to note that although the upper bound $N_Q^u(T)$ is smooth over all its domain, it is tight only when $\forall i \in [1, n] : \tau_i = 2^{-k}, k \in \mathbb{N}$. This difference between the objective function and $N_Q^u(T)$ means that it is possible to iteratively reduce $N_Q(W)$ by repeating the steps of the indirect method, described above, and improve our approximation of the optimum quantization.

Such an iterative algorithm would also help address the second challenge. Specifically, it allows us to use a simple model of the loss function (e.g. a linear or quadratic model) as a stand-in for the complex loss function. If in some iteration of the algorithm the model is inaccurate, it can be adjusted in following iteration. In our optimization algorithm, we will use a linear bound of loss function and adopt a *Trust Region* method to track its accuracy.

Trust region optimization methods iteratively optimize an approximation of the objective function. In order to guarantee the accuracy of this model, they further define a neighborhood around each point in the domain, called the trust region, inside which the model is a "sufficiently good" approximation of the objective function. The quality of the approximation is evaluated each iteration and the size of the trust region is updated accordingly. In our algorithm we adopt a similar technique, but apply it to the constraint instead of the objective function.

Lemma 2.2. *We refer to $m_l : R^n \to R$ as an accurate estimation of $l(W)$ in a subdomain of l like $D \subset R^n$, if:*

$$\forall W \in D : m_l(W) \leq \bar{l} \Rightarrow l(W) \leq \bar{l} \tag{13}$$

Let the radius Δ specify a spherical trust region around the point W_0 where the loss function l is accurately estimated by its first-order Taylor series expansion. Then, the constraint of equation **??**, *when $\|T\|_2 \leq \Delta$, is equivalent to:*

$$m_l^u(T) = l(W_0) + G^T T \leq \bar{l} \tag{14}$$

Where $G = [g_1 ... g_n]^T$ and $g_i = |[\nabla_W l]_i|, \forall i \in [1, n]$.

Proof. Since $\|T\|_2 \leq \Delta$, then ΔW as defined in equation **??** is inside the trust region. Therefore:

$$m_l^u(T) = \max_{\Delta W} \{l(W_0) + \nabla_W l(W_0)^T \Delta W\} \leq \bar{l} \tag{15}$$

□

Consequently, if we define such a trust region, we can simply the following problem as our subproblem in each iteration. We will present the solution for this subproblem in the next section.

$$\min_{T} N_Q^u(T) \tag{16}$$

$$m_l^u(T) \leq \bar{l} \tag{17}$$

Algorithm 2 summarizes the proposed method of solving the original optimization problem (equation 1). This algorithm first initializes the trust region radius Δ and the loss function bound $\bar{l}$. Then, it iteratively solves the subproblem of equation 17 and calculates the quantized parameters. If the loss function corresponding the quantized parameters violates the loss bound of equation 2, it means that the linear estimation was inaccurate over the trust region. Thus, we reduce the radius of the trust region and in the solve the subproblem over a smaller trust region in the next iteration. In this case, the calculated quantized parameters are rejected. Otherwise, we update the parameters. In addition, we grow the trust region unless if it grows larger than an upper bound on this radius ($\overline{Delta}$). We use two measures to examine convergence of the solution: the loss function and the trust region radius. If the loss function of the quantized model converges to the loss bound $\bar{l}$ or the trust region radius becomes very small, indicated by values ϵ and η, respectively. Note that the protocol to update the trust region radius as well as the trust region convergence condition used in this algorithm here are commonly used in trust region methods.

Algorithm 2 Adaptive Quantization

```
k ← 0
Initialize Δ, Δ̄ and l̄
while l̄ − l(W^k) ≥ ε and Δ ≥ η do
    Solve Trust Region Subproblem (Section 2.3)
    U = Quantize(W^k, T^k)
    if l(U) ≤ l̄ then
        W^{k+1} ← U
        Δ ← min(2Δ_k, Δ̄)
    else
        Δ_{k+1} ← (1/2) Δ_k
    end if
    k ← k + 1
end while
```

2.3 Trust Region Subproblem

We can directly solve the subproblem of equation **??** by writing its KKT conditions:

$$\nabla N_Q^u + \psi \nabla m_l^u = 0 \tag{18}$$

$$\psi(m_l^u - \bar{l}) = 0 \tag{19}$$

The gradient of the objective function, $\nabla N_Q^u = -\frac{1}{\ln 2}[\frac{1}{\tau_1} ... \frac{1}{\tau_n}]$ cannot be zero. Hence, $psi > 0$ and $m_l^u - \bar{l} = 0$.

$$\nabla N_Q^u + \psi \nabla m_l^u = 0 \Rightarrow \forall i \in [1, n] : -\frac{1}{\ln 2 \tau_i} + \psi g_i \Rightarrow \forall i \in [1, n] : \tau_i g_i = \frac{1}{\psi \ln 2} \Rightarrow G^T T = \frac{n}{\psi \ln 2}$$

$$m_l^u - \bar{l} = 0 \Rightarrow l(W) + \frac{n}{\psi \ln 2} - \bar{l} = 0 \Rightarrow \frac{n}{\psi \ln 2} = \frac{\bar{l} - l(W)}{n} \rightarrow \forall i \in [1, n] : \tau_i = \frac{\bar{l} - l(W)}{n g_i}$$

If for the resulting tolerance vector $\|T\| > \Delta$, we scale T so that its norm would be equal to Δ.

This solution is correct only when $g_i > 0$ for all i. It is possible, however, that there exists some i for which $g_i = 0$. In this case, we use the following equation to calculate the optimal solution T^*.

$$\tau_i^* = \begin{cases} \frac{d}{ng_i} & g_i > 0 \\ |\omega_i| & g_i = 0 \end{cases} \tag{20}$$

We treat singular points this way for three reasons. First, a gradient of zero with respect to ω_i means that the cost is not very sensitive to its value. Thus, it might be possible to eliminate it quickly. Second, this insensitivity of the cost to ω_i means that large deviations in its value would not significantly affect the cost. Finally, setting the value of τ_i^* to a large value relative to other tolerances reduces their values after normalization to the trust region radius. Thus, the effect of a singularity on other parameters is reduced. After finding tolerance values, we once again calculate T^k by normalizing T^* to the trust region radius.

2.4 Choosing Hyper-parameters

The hyper-parameters, $\overline{\Delta}$, and $\bar{l}$, determin the speed of convergence and the final classification accuracy. Smaller trust regions result in slower convergence, while larger trust regions produce higher error rates. For $\overline{\Delta}$, since remaining values could ultimately be represented by 0 bits (pruned), we choose $2^0\sqrt{n}$. Similarly, the higher the loss bound $\bar{l}$, the lower the accuracy, while small values of loss bound prevent effective model size reduction. We choose this value in our algorithm on the flye. That is, we start off by conservatively choosing a small value for $\bar{l}$ (e.g. $\bar{l} = l(W_0)$) and then increase it everytime the algorithm 2 converges. Algorithm 3 provides the details of this process. At the begining the loss bound is chosen to be the loss of the floating-point trained model. After quantizing the model using this bound, the bound value is increased by $Scale$. These steps are repeated $Steps$ times. In our experiments in the next section $Scale$ and $Steps$ are set to 1.1 and 20, respectively.

Algorithm 3 Adaptive Quantization

Initialize $Steps$ and $Scale$
$\bar{l} \leftarrow l(W_0)$
while $Steps > 0$ **do**
  Quantize model using Algorithm 2.
  $Steps \leftarrow Steps - 1$
  $\bar{l} \leftarrow \bar{l} * Scale$
end while

3 Evaluation and Discussion

We evaluate the performance of the proposed quantization method in compressing trained NN models using a set of classification datasets. In our evaluation, we train three networks using three common datasets. Then, we use the adaptive quantization method to quantize these trained models. In order to achieve the smallest model sizes, we apply the quantization method three times and after each time we retrain the network for one epoch in the full precision domain. We will experimentally show that three passes of quantization is sufficient for achieving good accuracy as well as model size reduction. In this section, we discuss the benchmarks used in these experiments and analyze their quantization results.

3.1 Benchmarks

We use MNIST (LeCun et al. 1998a), CIFAR-10 (Krizhevsky and Hinton 2009), and SVHN (Netzer et al. 2011) in our experiments. MNIST is a small dataset containing 28×28 images of handwritten digits from 0 to 9. For this dataset, we use the LeNet-5 network (LeCun et al. 1998b). For both CIFAR-10, a dataset of 32×32 images in 10 classes, and SVHN, a large dataset of 32×32 real-world images of digits, we employ the same organization of convolution layers and fully connected layers as networks used in BNN (Hubara et al. 2016b). These networks are based on VGG (Simonyan and Zisserman 2014) but have parameters in full precision. The only difference is that in the case of

CIFAR-10, we use 4096 neurons instead of the 1024 neurons used in BNN, and we do not use batch normalization layers. We train these models using a Cross entropy loss function. For CIFAR-10, we also add an l2 regularization term. Table 1 shows the specifications of the baseline trained models.

Table 1: Baseline models

Dataset	Model	Model Size	Error Rate
MNIST	LeNet-5 (LeCun et al. 1998b)	$12Mb$	0.65%
CIFAR-10	(Hubara et al. 2016b)	$612Mb$	9.08%
SVHN	(Hubara et al. 2016b)	$110Mb$	7.26%

3.2 Execution Time

The key contributors to the computational load of the proposed technique are back propagation (to calculate gradients) and quantization (algorithm 2). Both these operations can be completed in $O(n)$ complexity. We implement the proposed quantization on Intel Core i7 CPU (3.5 GHz) with Titan X GPU performing training and quantization. The timing results of the algorithm have been summarized in table 2.

Table 2: Timining results for training and quantization of the benchmark models in **seconds**

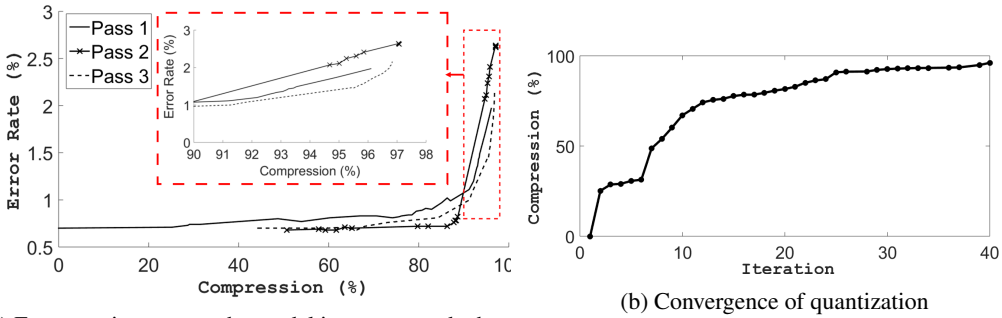
Dataset	Training	Quantization
MNIST	120	300
CIFAR-10	4560	4320
SVHN	1920	2580

3.3 Quantization Results

We evaluate the performance of the quantization algorithm by analyzing the compression rate of the models and their respective error rates after each pass of quantization without retraining. As discussed in section 2, the quantization algorithm will try to reduce parameter precision while maintaining a minimum classification accuracy loss. Here we present the results of these experiments.

Figure 1a depicts the error rate of LeNet trained on the MNIST dataset as it is compressed through three passes of adaptive quantization and retraining. As this figure shows, the second and third passes tend to have smaller error rates compared to the first pass. But, at the end of the second pass the error rate is higher than the first one. This can be improved by increasing the number of epochs for retraining or choosing a lower cost bound ($\overline{C}$). By the end of the third pass, the highest compression rate is achieved. However, due to its small difference in compression rate compared to its preceding pass, we do not expect to achieve more improvements by continuing the retraining process.

We also evaluate the convergence speed of the algorithm by measuring the compression rate ($1 - \frac{sizeofquantizedmodel}{sizeoforiginalmodel}$) of the model after each iteration. The results of this experiment for one pass of quantization have been depicted in Figure 1b. As shown in the figure, the size of the model is reduced quickly during the initial iterations. This portion of the experiment corresponds to the part of Figure 1a where the error rate is constant. However, as quantization continues, the model experiences diminishing returns in size reduction. After 25 iterations, little reduction in model size is achieved. The lower density of data points past this point is due to the generated steps failing the test on the upper bound on cost in Algorithm 2. Consequently, the algorithm reduces the trust region size and recalculates the tolerance steps.



(a) Error rate increase as the model is compressed when retraining is applied

(b) Convergence of quantization

Figure 1: Quantization of LeNet-5 model trained on MNIST dataset

4 Comparison with Related Approaches

The generality of the proposed technique makes it adaptable to other model compression techniques. In this section, we review some of the most notable of these techniques and examine the relationship between their approach and ours. Specifically, we will explain how the proposed approach subsumes previous ones and how it can be specialized to implement them in an improved way.

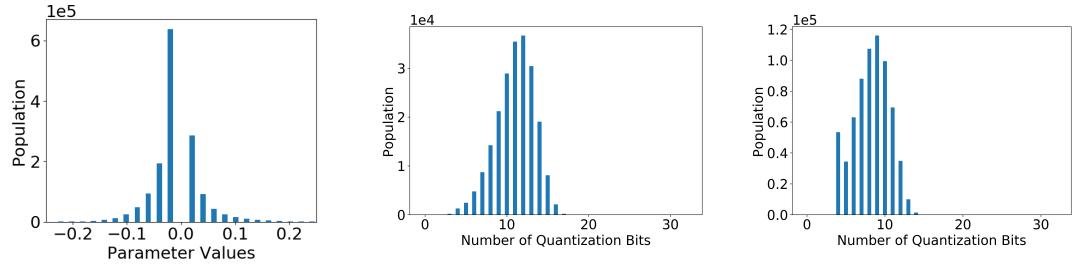
Pruning: In pruning, small parameters in a trained network are set to zero, which means that their corresponding connections are eliminated from the network. Han et al. showed that by using pruning, the number of connections in the network can be reduced substantially (Han et al. 2015). Adaptive quantization confirms similar observations that in a NN, most connections can be eliminated. In fact, adaptive quantization eliminates $5.6\times$, $5.6\times$, and $5.3\times$ of the connections in the networks trained for MNIST, CFAR, and SVHN, respectively. These number are lower compared to their corresponding values achieved using deep compression. The reason for this difference is that deep compression amortizes for the loss of accuracy due to pruning of a large number of parameters by using full precision in the remaining ones. In contrast, adaptive quantization eliminates fewer connections and instead quantizes the remaining parameters.

While adaptive quantization identifies connections that can be eliminated automatically, deep compression identifies them by setting parameters below a certain threshold to zero. However, this technique might not be suitable in some cases. For example, we consider the network model trained for CIFAR-10. This network has been trained using $L2$ regularization, which helps avoid over fitting. As a result, the trained model has a large population of small-valued parameters. Looking at Figure 2a which shows the distribution of parameter values in the CIFAR-10 model, we see that most populate a small range around zero. Such cases can make choosing a good threshold for pruning difficult.

Weight Sharing: The goal of weight sharing is to create a small dictionary of parameters. The parameters are grouped into bins whose members will all have the same value. Therefore, instead of storing parameters themselves, the weights can be replaced by the index in the dictionary corresponding to that parameter's bin. Deep compression implements weight sharing by applying k-means clustering to trained network parameters (Han et al. 2015). Similarly, Smaragh et al. implements weight sharing by iteratively applying k-means clustering and retraining in order to find the best dictionary (Sarmagh et al. 2017). In both of these works, the number of dictionary entries are fixed in advance. Adaptive quantization, on the other hand, produces the bins as a byproduct and does not make assumptions on the total number of bins.

Deep compression also identifies that the accuracy of the network is less sensitive to the fully connected layers compared to the convolution layers. Because of that it allocates a smaller dictionary for storing them. Looking at figures 2b and 2c, we can see that results of adaptive quantization are consistent with this observation. These figures show the distribution of quantization bits for one convolution layer and one fully connected layer in the quantized CIFAR-10 network. It is clear from these figures that parameters in the fully connected layer, on average require fewer quantization bits.

Binarization and Quantization: Binarized Neural Networks (Courbariaux et al. 2015; Hubara et al. 2016b) and Quantized Neural Networks (Hubara et al. 2016a) can reduce model size by assuming the same quantization precision for all parameters in the network or all parameters in a layer.

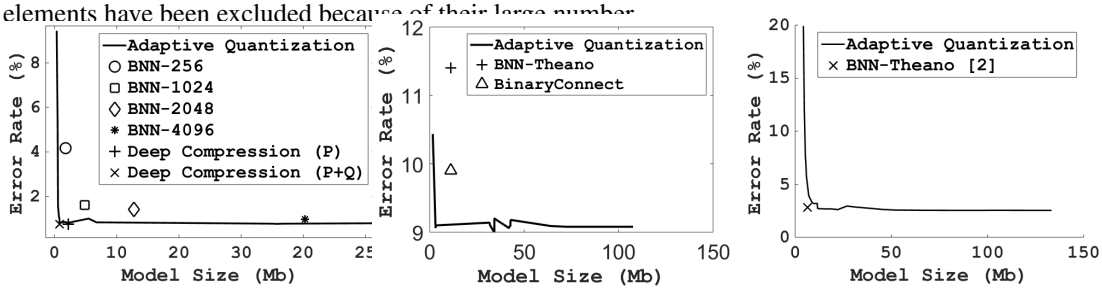


(a) Distribution of parameter values.

(b) Distribution of Quantization bits in the first convolution layer

(c) Distribution of Quantization bits in the first fully connected layer

Figure 2: Distributions of parameter values and quantization bits for the CIFAR-10 network. Zero elements have been excluded because of their large number.



(a) MNIST

(b) CIFAR-10

(c) SVHN

Figure 3: Trade-off between accuracy and error rate for benchmark datasets. BNN-2048, BNN-24096, BNN-Theano are from Hubara et al. (2016b); BinaryConnect is from Courbariaux et al. (2015); and Deep Compression is from Han et al. (2015)

These works often design quantized training algorithms to maintain the same parameter precisions throughout training. These techniques can be slower than full-precision training. In contrast, adaptive quantization allows for unique quantization precisions for *each parameter*. This way, for a trained model, it can find a notably smaller network, indicating the limits of quantization. In addition, it does not require quantized training.

In order to compare adaptive quantization with previous works, we quantize trained models with different training cost bounds. This way, for each dataset, we produce several models with different sizes and error rates. We illustrate the results of these experiments in Figure 3, showing the trade-off between model size and error rate.

In both MNIST and CIFAR-10, adaptive quantization produces curves below the ones achieved in BNN (Hubara et al. 2016b) and BinaryConnect (Courbariaux et al. 2015), and it shows a smaller model size while maintaining the same error rate or the same model size with a smaller error rate. In the case of SVHN, the BNN achieves a better result compared to adaptive quantization. This is due in part to the error rate of the full-precision model being higher than the BNN model.

Comparison of adaptive quantization against deep compression (when pruning is applied) for the MNIST set shows some similarity between the two techniques, although deep compression achieves a slightly smaller error rate for the same model size. This advantage comes from two differences between the two techniques. First, deep compression uses full precision, floating point parameters while adaptive quantization uses quantized parameters, which requires less computational complexity (fixed-point operation vs. floating-point operation). Second, after pruning the network, deep compression performs a complete retraining of the network. In contrast, adaptive quantization performs little retraining.

Furthermore, adaptive quantization can be used to identify groups of parameters that need to be represented with more precision. In a more general sense, the flexibility of adaptive quantization allows it to be specialized for more constrained forms of quantization. For example, if the quantization requires that all parameters in the same layer have the same quantization precision, adaptive quantization could find the best model. This could be implemented by solving the same minimization problem as in Equation **??**, except the length of T would be equal to the number of layers. In this case, g_i in

Equation **??** would be $\sum_{j \in J_i} |\frac{\partial C}{\partial \omega_j}(\omega_j^k)|$, where J_i is the set of parameter indices belonging to layer i. The rest of the solution would be the same.

5 Conclusion

In this work, we quantize neural network models such that only parameters critical to the accuracy are represented with high accuracy. The goal is to minimize data movement and simplify computations needed for inference in order to accelerate implementations on resource constrained hardware. To achieve acceleration, the proposed technique prunes unnecessary parameters or reduces their precision. Combined with existing fixed-point computation techniques such as SWAR or Bit-pragmatic computation , we expect these small fixed-point models to achieve fast inference with high accuracy. We have confirmed the effectiveness of this technique through experiments on several benchmarks. Through this technique, our experiments show, DNN model sizes can be reduced significantly without loss of accuracy. The resulting models are significantly smaller than state-of-the-art quantization technique. Furthermore, it can provide similar results to floating-point model compression techniques.

References

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

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

J. Devlin, R. Zbib, Z. Huang, T. Lamar, R. M. Schwartz, and J. Makhoul, "Fast and robust neural network joint models for statistical machine translation." in *ACL (1)*. Citeseer, 2014, pp. 1370–1380.

M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

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

L. Zhao, S. Liao, Y. Wang, J. Tang, and B. Yuan, "Theoretical properties for neural networks with weight matrices of low displacement rank," *arXiv preprint arXiv:1703.00144*, 2017.

N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 16–25.

J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 26–35.

I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *arXiv preprint arXiv:1609.07061*, 2016.

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

I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 4107–4115.

Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," 1998.

A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.

Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2, 2011, p. 5.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

M. Sarmagh, M. Ghasemzadeh, and F. Koushanfar, "Customizing neural networks for efficient fpga implementation," *The 25th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM 2017)*, 2017.