ADAPTIVE QUANTIZATION OF NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

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 & 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., 2016b; Zhou et al., 2016; Hubara et al., 2016a; Courbariaux et al., 2015).

Although these quantization methods can significantly reduce model complexity, they generally have two key constraints. First, they ignore the accuracy degradation resulting from quantization, during the quantization, and tend to remedy it, separately, through quantized learning schemes. However, such schemes have the disadvantage of converging very slowly compared to full-precision learning methods. Second, they treat all network parameters similarly and assign them the same quantization depth. This is while previous works (Hubara et al., 2016a;b; Han et al., 2015) have shown different parameters do not contribute to the model accuracy equally. Disregarding this variation limits the maximum achievable 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 searching for, in particular, redundant filters (DenseNets (Huang et al., 2016)) and redundant connections (Deep Compression (Han et al., 2015)). 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 (Han et al., 2015) 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.

It is also important to discuss the benefits of this fine-grained quantization for performance. In particular, besides its storage advantages, we argue that this method can reduce the computation, if the target hardware can take advantage of the non-standard, yet small quantization depths that our approach produces. We note that there exist techniques on CPU and GPU for fixed-point arithmetics with non-standard quantization widths (e.g. SWAR (Cameron & Lin, 2009)). But, we believe our proposed quantization is ideal for FPGAs and specialized hardware. The flexibility of these platforms allows for design of efficient computation units that directly process the quantized parameters. This was recently explored by Albericio et al. (2016), who designed specialized computation units for variable bit width parameters that eliminated ineffectual computations. By minimizing the overall number of bits that need to be processed for a network, the proposed quantization achieves the same effect.

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. In the lemma below we derive such a bound. To our knowledge, this is the first time such a bound has been developed in the literature.

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:*

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

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

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

Algorithm 1 estimates ω_i with a rounding error of up to $\frac{1}{2^{N_q+1}}$. Thus, for the worst case to be consistent with the constraint of equation 4, we need:

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

In this case the minimization guarantees that the smallest $N_q(\omega_i)$ is chosen. This entails:

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

In general, we can expect $N_q(\omega_i)$ to be smaller than this worst case. Consequently:

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

□

We should note that for simplicity algorithm 1 assumes that parameters ω_i are unsigned and in the range $[0, 1]$. In practice, for signed values, we quantize the absolute value using the same method, and use one bit to represent the sign. For models with parameters in a range larger than $[-1, 1]$, say $[-r, r]$ with $r > 1$, we perform the quantization similarly. The only difference is that we first scale the parameters and their tolerances by $\frac{1}{r}$ to bring them to $[-1, 1]$ range and then quantize the parameters. When computing the output of a quantized layer, then, we multiply the result by r to account for the scale. Following this, the same equations as above will hold.

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, such that 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 can be 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 the 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 11, 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 12 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 use 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 11 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 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, and we enlarge the trust region unless if it grows larger than an upper bound like $\overline{\Delta}$. 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)
    U = Quantize(W^k, T^k)
    if l(U) ≤ l̄ then
        W^{k+1} ← U
        Δ_{k+1} ← 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 11 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 $m_l^u - \bar{l} = 0$.

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

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

For the resulting tolerance vector $\|T\|_2 > \Delta$, 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^*.

$$\tau_i^* = \begin{cases} \frac{\bar{l} - l(W^k)}{n g_i} & g_i > 0 \\ |\omega_i| & g_i = 0 \end{cases} \tag{20}$$

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 parameter values. 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 the Hyper-parameters

The hyper-parameters, $\overline{\Delta}$, and $\bar{l}$, determine 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 fly. 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 every time algorithm 2 converges. Algorithm 3 provides the details of this process. At the beginning 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 adaptive quantization on three popular image classification benchmarks. For each, we first train a neural network in the floating-point domain, and then apply a pass of algorithm

3 to compress the trained model. In both these steps, we use the same batchsize to calculate the gradients and update the parameters. To further reduce the model size, we tune the accuracy of the quantized model in the floating-point domain and quantize the tuned model by reapplying a pass of algorithm 3. For each benchmark, we repeat this process three times, and experimentally show that this produces the smallest model. In the end we evaluate the accuracy and the size of the quantized model. Specifically, we determine the overall number of bits (quantization bits and the sign bits), and evaluate how much reduction in the model size has been achieved.

We note that it is also important to evaluate the potential overheads of bookkeeping for the quantization depths. However, we should have in mind that bookkeeping has an intricate relationship with the target hardware, which may lead to radically different results on different hardware platforms (ranging from 0 to 60%). For example, our experiments show that on specialized hardware, such as the one designed by Albericio et al (2017) for processing variable bit width CNN, we can fully offset all bookkeeping overheads of storing quantization depths, while CPU/GPU may require up to 60% additional storage. We will study this complex relationship separately, in our future work, and in the context of hardware implementation. In this paper, we limit the scope to algorithm analysis, independent of the underlying hardware architectures.

3.1 Benchmarks

We use MNIST (LeCun et al., 1998b), CIFAR-10 (Krizhevsky & 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., 1998a). 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., 2016a). These networks are based on VGG (Simonyan & 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 L_2 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., 1998a)	$12Mb$	0.65%
CIFAR-10	(Hubara et al., 2016a)	$612Mb$	9.08%
SVHN	(Hubara et al., 2016a)	$110Mb$	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.

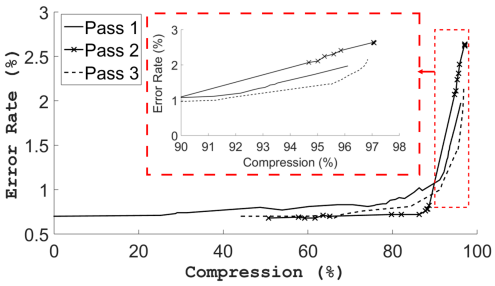
3.3 Quantization Results

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 precisions while maintaining a minimum classification accuracy loss. Here we present the results of these experiments.

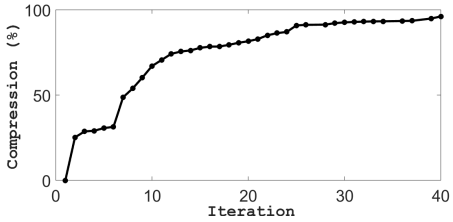
Figure 1a depicts the error rate of LeNet trained on the MNIST dataset as it is compressed through three passes of adaptive quantization and retraining. As this figure shows, the second and third passes

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

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



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



(b) Convergence of quantization. Iterations refers to iterations of the loop in algorithm 2.

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

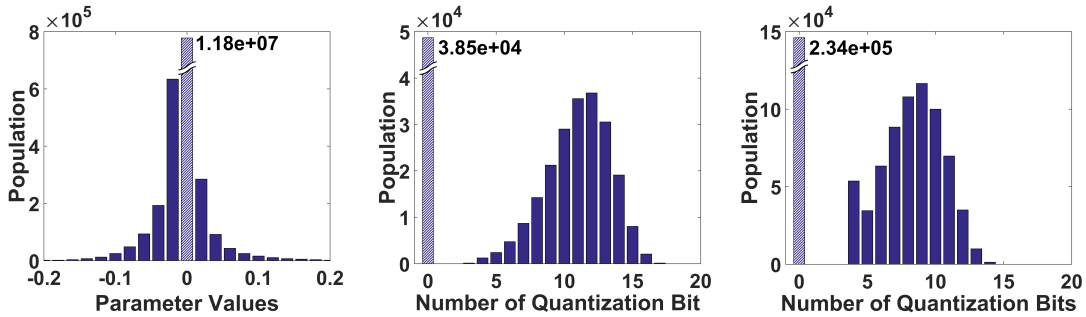
tend to have smaller error rates compared to the first pass. But, at the end of the second pass the error rate is higher than the first one. This can be improved by increasing the number of epochs for retraining or choosing a lower cost bound ($\bar{l}$). By the end of the third pass, the highest compression 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 ($1 - \frac{\textit{size of quantized model}}{\textit{size of original model}}$) of the model after each iteration of the loop in algorithm 2. The results of this experiment for one pass of quantization of algorithm 3 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.

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. (2015) showed that by using pruning, the number of connections in the network can be reduced substantially. Adaptive quantization confirms similar observations that in a DNN, 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.



(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. Pruned parameters have been depicted using a hatch pattern and their population has been reported besides their corresponding bar.

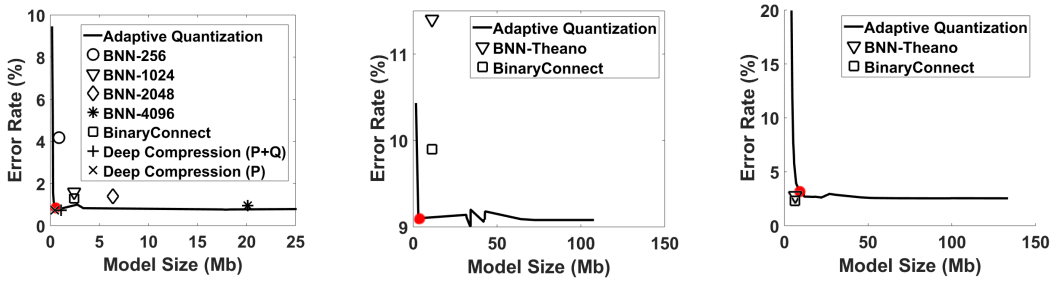
While adaptive quantization identifies connections that can be eliminated automatically, deep compression identifies them by setting parameters below a certain threshold to zero. However, this technique might not be suitable in some cases. For example, we consider the network model trained for CIFAR-10. This network has been trained using L_2 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 (Han et al., 2015) implements weight sharing by applying k-means clustering to trained network parameters. Similarly, Sarmagh et al. (2017) implement weight sharing by iteratively applying k-means clustering and retraining in order to find the best dictionary . 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 Figure 2b and Figure 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., 2016a) and Quantized Neural Networks (Hubara et al., 2016b) can reduce model size by assuming the same quantization precision for all parameters in the network or all parameters in a layer. In the extreme case of BNNs, parameters are quantized to only 1 bit. In contrast, our experiments show that through pruning and quantization, the proposed approach quantizes parameters of the networks for MNIST, CIFAR-10, and SVHN by equivalent of 0.03, 0.27, and 1.3 bits per parameter with 0.12%, -0.02%, and 0.7% decrease in accuracy, respectively. Thus, our approach produces competative quantizations. Furthermore, previous quantization techniques 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.

Next, we compare adaptive quantization with previous works on compressing neural network models, in reducing the model size. The results for these comparisons have been presented in Figure 3, which



(a) MNIST (b) CIFAR-10 (c) SVHN

Figure 3: Trade-off between accuracy and error rate for benchmark datasets. The optimal points for MNIST, CIFAR-10, and SVHN (highlighted red) achieve $64\times$, $35\times$, and $14\times$ compression and correspond to 0.12%, -0.02%, and 0.7% decrease in accuracy, respectively. BNN-2048, BNN-24096, BNN-Theano are from (Hubara et al., 2016a); BinaryConnect is from (Courbariaux et al., 2015); and Deep Compression is from (Han et al., 2015)

shows the trade-offs that adaptive quantization offers between accuracy and model size. As this figure shows, adaptive quantization in many cases out performs previous methods and consistently produces compressions better than or comparable to state-of-the-art. In particular, we mark an optimal trade-off for each model in Figure 3 (red highlight). In these points, the proposed method achieves $64\times$, $35\times$, and $14\times$ compression and correspond to 0.12%, -0.02%, and 0.7% decrease in accuracy, respectively. This always improves or is comparable to the state-of-the-art of Quantization (BNN and BinaryConnect) and Pruning and Weight-Sharing (Deep Compression). Below, we discuss these trade-offs in more details.

In both MNIST and CIFAR-10, adaptive quantization produces curves below the ones achieved in BNN (Hubara et al., 2016a) 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 slightly 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 11, except the length of T would be equal to the number of layers. In this case, g_i in Equation 2.3 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 (Cameron & Lin, 2009) or Bit-pragmatic computation (Albericio et al., 2016), 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

Jorge Albericio, Patrick Judd, Alberto Delmas Lascorz, Sayeh Sharify, and Andreas Moshovos. Bit-pragmatic deep neural network computing. *CoRR*, abs/1610.06920, 2016. URL `http://arxiv.org/abs/1610.06920`.

Robert D. Cameron and Dan Lin. Architectural support for swar text processing with parallel bit streams: The inductive doubling principle. *SIGPLAN Not.*, 44(3):337–348, March 2009. ISSN 0362-1340. doi: 10.1145/1508284.1508283. URL `http://doi.acm.org/10.1145/1508284.1508283`.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pp. 3123–3131, 2015.

Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M Schwartz, and John Makhoul. Fast and robust neural network joint models for statistical machine translation. In *ACL (1)*, pp. 1370–1380. Citeseer, 2014.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems*, pp. 4107–4115, 2016a.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016b.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998a.

Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998b.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 5, 2011.

Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen 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*, pp. 26–35. ACM, 2016.

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

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

Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu 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*, pp. 16–25. ACM, 2016.

Liang Zhao, Siyu Liao, Yanzhi Wang, Jian Tang, and Bo Yuan. Theoretical properties for neural networks with weight matrices of low displacement rank. *arXiv preprint arXiv:1703.00144*, 2017.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.