

Balanced Quantization: An Effective and Efficient Approach to Quantized Neural Networks

Shuchang Zhou¹²³, Yuzhi Wang³, He Wen³, Qinyao He³ and Yuheng Zou³

¹University of Chinese Academy of Sciences,
Beijing 100049, China

²State Key Laboratory of Computer Architecture,
Institute of Computing Technology,
Chinese Academy of Sciences,
Beijing 100190, China

³ Megvii Inc., Beijing 100190, China

shuchang.zhou@gmail.com, yz-wang12@mails.tsinghua.edu.cn,
{wenhe,hqy,zouyuheng}@megvii.com

December 20, 2016; revised Mar. 5, 2017

Abstract

Quantized Neural Networks (QNNs), which use low bitwidth numbers for representing parameters and performing computations, have been proposed to reduce the computation complexity, storage size and memory usage. In QNNs, parameters and activations are uniformly quantized, such that the multiplications and additions can be accelerated by bitwise operations. However, distributions of parameters in Neural Networks are often imbalanced, such that the uniform quantization determined from extremal values may under utilize available bitwidth. In this paper, we propose a novel quantization method that can ensure the balance of distributions of quantized values. Our method first recursively partitions the parameters by percentiles into balanced bins, and then applies uniform quantization. We also introduce computationally cheaper approximations of percentiles to reduce the computation overhead introduced. Overall, our method improves the prediction accuracies of QNNs without introducing extra computation during inference, has negligible impact on training speed, and is applicable to both Convolutional Neural Networks and Recurrent Neural Networks. Experiments on standard datasets including ImageNet and Penn Treebank confirm the effectiveness of our method. On Ima-

geNet, the top-5 error rate of our 4-bit quantized GoogLeNet model is 12.7%, which is superior to the state-of-the-arts of QNNs.

1 Introduction

Deep Neural Networks (DNNs) have attracted considerable research interests over the past decade. In various applications, including computer vision [1, 2, 3, 4], speech recognition [5, 6], natural language processing [7, 8, 9], and computer games [10, 11], DNNs have demonstrated their ability to model nonlinear relationships in massive amount of data and their robustness to realworld noise. However, the modeling capacities of DNNs are roughly proportional to their computational complexity and number of parameters [12]. Hence many DNNs, like VGGNet [13], GoogLeNet [14] and ResNet [15], which are widely used in computer vision applications, require billions of multiply-accumulate operations (MACs) even for an input image of width and height of 224. Moreover, as these DNN models use many channels of activations (feature maps) for intermediate representations, they have a large runtime memory footprint and storage size. Such vast amount of resource requirement impedes the adoption of DNNs on devices with limited computation resource and power supply [16], and in user-interactive scenarios where instant responses are expected. A similar argument also applies to Recurrent Neural Networks (RNNs). In particular, the transition and embedding matrices in a Long Short Time Memory (LSTM) [17] or a Gated Recurrent Units (GRU) [18] model have dense connections that make them particularly demanding in both computation and storage.

Many approaches have been proposed to accelerate the computation or reduce the memory footprint and storage size of DNNs. One approach from the hardware perspective is designing hardware accelerators for the computationally expensive operations in DNNs [19, 20, 21]. From the algorithmic perspective, a popular route to faster and smaller models is to impose constraints on the parameters of a DNN to reduce the number of free parameters and computational complexity, like low-rankness [22, 23, 24, 25, 26, 27], sparsity [28, 29, 30, 31], circulant property [32], and sharing of weights [33, 34]. However, these methods use high bitwidth numbers for computations in general, which requires availability of high precision MAC instructions that incur high hardware complexity [35]. In contrast, several previous works have demonstrated that low bitwidth numbers may be sufficient for performing inferences with DNNs. For example, in [36, 37, 38], trained DNNs are quantized to use 8-bit numbers for storing parameters and performing

computations, without incurring significant degradation of prediction quality. Gong *et al.* [39] also applied vector quantization to speed up inferences of DNNs. However, these works [36, 37, 38, 39] did not integrate the quantization operations into the training process of a DNNs, as the discrete quantized values necessarily would have zero gradients, which would break the Back-Propagation (BP) algorithm. Applying quantization as a post-processing step is far from satisfactory as the quantized DNNs do not have a chance to adapt to the quantization errors [40]. Consequently, 8-bit was generally taken to be a limit for post-training quantization of DNNs [41].

Recently, Quantized Neural Networks (QNNs) [42, 43, 44, 45, 46] have been proposed to further reduce the bitwidths of DNNs, by incorporating quantization into the training process. The key enabling technique is a trick called Straight Through Estimator (STE) [47, 48, 49], which is based on the following observation: as the quantized value is an approximation of the original value, we can substitute the gradient with respect to the quantized value for the gradient of original value. Simple as it is, the trick allows the inclusion of quantization into the computation graph of BP and allows QNNs to represent parameters, activations and gradients with low bitwidth numbers. The QNN technique has been successfully applied to both CNNs and RNNs [50, 51], to successfully produce lower bitwidth versions of AlexNet, ResNet-18 and GoogLeNet that have comparable prediction accuracies as their floating point counterparts. However, the degradation of prediction accuracy is still significant for most QNNs, especially when quantizing to less than 4-bit [52, 53].

In this paper, we propose a Balanced Quantization method that improves the prediction accuracies of QNNs. In general, QNNs employ uniform quantization to eliminate floating point operations during the inference process by exploiting the bitwise operations. However, the parameters of neural networks often have a bell-shaped distribution and sporadic large outliers, making the quantized values not evenly distributed among possible values when uniform quantization is applied. In the extreme case, some of the possible quantized values are never used. To remedy this, we propose to use a novel quantization method that ensures the balanced distribution of quantized values.

This paper makes the following contributions:

1. We propose a Balanced Quantization method for the quantization of parameters of QNNs. The method emphasizes on producing balanced distributions of quantized values rather than preserving extremal values, by using percentiles as quantization thresholds. As a result, effec-

tive bitwidths of quantized models are increased. (See Subsection 3.2)

2. To reduce the computation overhead introduced by computing percentiles, we approximate medians by means, which are computationally more efficient on existing hardware. The efficacy of the approximation is empirically validated. (see Subsection 3.3)
3. Experiments confirm that our method significantly improves the prediction accuracies of CNNs and RNNs on standard datasets like ImageNet and Penn Treebank. (see Section 4)
4. The implementation of Balanced Quantization will be available online, in TensorFlow [54] framework.

2 Quantized Neural Networks

In this section we introduce the notations and algorithms of QNNs. We also show how QNNs can exploit bitwise operations for speeding up computations and how to incorporate quantization steps into computation graphs of QNNs during training.

2.1 Notations

We will use the rounding operation intensively in this paper. For tie-breaking, we apply the “round half towards zero” rule, which rounds positive numbers with fraction $\frac{1}{2}$ down and negative numbers with fraction $\frac{1}{2}$ up. We assign the name of “round-to-zero” for this variant of rounding:

$$\textit{round-to-zero}(x) \stackrel{\text{def}}{=} \text{sgn}(x) \left\lfloor |x| - \frac{1}{2} \right\rfloor.$$

Without loss of generality, we represent weight parameters of a neural network as a matrix \mathbf{W} . When doing k -bit uniform quantization with the step length $\frac{1}{2^{k-1}}$, we can define a utility function Q_k that converts floating point numbers in close interval $[0, 1]$ to fixed point numbers as follows: .

$$Q_k(\mathbf{W}) \stackrel{\text{def}}{=} \frac{\textit{round-to-zero}((2^k - 1)\mathbf{W})}{2^k - 1},$$

$$0 \leq w_{i,j} \leq 1 \forall i, j. \tag{1}$$

The outputs of Q_k are the fixed point values $0, \frac{1}{2^{k-1}}, \frac{2}{2^{k-1}}, \dots, 1$.

In a Quantized Neural Network, we use Q_k for the quantization of parameters, activation and gradients. When quantizing parameters, as the utility function Q_k requires the input to be in close interval $[0, 1]$, we should first map the parameters \mathbf{W} to that value range. The method in [51, 53] uses the following affine transform to change the value range:

Definition 1 (k -bit Uniform Quantization)

$$\begin{aligned} \varphi(\mathbf{W}) &\stackrel{\text{def}}{=} \frac{\mathbf{W}}{2\max(|\mathbf{W}|)} + \frac{1}{2}, \\ \text{quant}_k(\mathbf{W}) &\stackrel{\text{def}}{=} \varphi^{-1}(Q_k(\varphi(\mathbf{W}))), \end{aligned}$$

where the subscript k in quant_k stands for k -bit quantization, and $|\mathbf{W}|$ is a matrix with values being the absolute values of corresponding entries in \mathbf{W} .

As $-\max(|\mathbf{W}|) \leq w_{i,j} \leq \max(|\mathbf{W}|)$, we have $0 \leq \frac{w_{i,j}}{2\max(|\mathbf{W}|)} + \frac{1}{2} \leq 1$. We can then apply Q_k to get the fixed point values $Q_k(\varphi(\mathbf{W}))$, which are affine transformed by φ^{-1} to restore the value range back to the closed interval $[-\max(|\mathbf{W}|), \max(|\mathbf{W}|)]$.

2.2 Simplistic View of Quantized Neural Network

QNNs are Neural Networks that use quantized values for computations. Because the convolutions between inputs and convolution kernels can also be represented as matrix products, w.l.o.g., we take a Multi-Layer Perceptron (MLP) as an example through out the rest of this paper.

Let the outputs, activation function, weight parameters and bias parameters of the i -th layer of a neural network be \mathbf{X}_i , σ_i , \mathbf{W}_i and \mathbf{b}_i , respectively. The i -th Convolution/Fully-Connected layer can be represented as:

$$\mathbf{X}_i = \sigma_i(\mathbf{W}_i \mathbf{X}_{i-1} + \mathbf{b}_i).$$

The corresponding formula for the i -th Convolution/Fully-Connected layer of a QNN is:

$$\begin{aligned} \mathbf{X}_i^q &= Q_A(\sigma_i(\mathbf{W}_i^q \mathbf{X}_{i-1}^q + \mathbf{b}_i)) \\ \mathbf{W}_i^q &= Q_W(\mathbf{W}_i), \end{aligned} \tag{2}$$

where \mathbf{W}_i^q and \mathbf{X}_i^q are quantized weights and activations respectively; Q_W and Q_A are quantization functions. Note the bias parameters \mathbf{b}_i may not need be quantized, for reasons we will explain in Appendix A.

Other types of layers like pooling layers may also take quantized values as inputs and outputs. The input to the first layer of a QNN may have higher bitwidth than the rest of the network to preserve information [51].

2.3 Exploiting Bitwise Operations in QNN

Using quantized values for computation makes it possible to use fixed point operations instead of floating point operations. We next show how to perform dot products between quantized numbers by bitwise operations.

We first consider the dot products between k -bit fixed point numbers. In the extreme case of $k = 1$, dot products are done between bit strings, which allows for the following method of using bitwise operations:

$$\mathbf{x} \cdot \mathbf{y} = \text{bitcount}(\text{and}(\mathbf{x}, \mathbf{y})), \forall i, x_i, y_i \in \{0, 1\},$$

where “bitcount” counts the number of 1 in a bit string, and “and” performs bitwise AND operation.

In the multi-bit case ($k > 1$), we may also exploit the above kernel as in [42]. Assume \mathbf{x} is a sequence of M -bit fixed point integers s.t. $\mathbf{x} = \sum_{m=0}^{M-1} c_m(\mathbf{x})2^m$ and \mathbf{y} is a sequence of K -bit fixed point integers s.t. $\mathbf{y} = \sum_{k=0}^{K-1} c_k(\mathbf{y})2^k$ where $(c_m(\mathbf{x}))_{m=0}^{M-1}$ and $(c_k(\mathbf{y}))_{k=0}^{K-1}$ are bit vectors, the dot product of \mathbf{x} and \mathbf{y} can be computed by bitwise operations as:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} 2^{m+k} \text{bitcount}[\text{and}(c_m(\mathbf{x}), c_k(\mathbf{y}))],$$

$$c_m(\mathbf{x})_i, c_k(\mathbf{y})_i \in \{0, 1\} \forall i, m, k.$$

In the above equation, the computation complexity is $O(MK)$, i.e., directly proportional to the product of bitwidths of \mathbf{x} and \mathbf{y} . Hence it is beneficial to reduce the bitwidth of a QNN as long as the prediction accuracy is kept at the same level. It has been demonstrated that exploiting dot-product kernels allows for efficient software [51] and hardware implementations [55, 56].

In Formula 2, the matrix multiplication happens between the quantized values \mathbf{W}_i^q and \mathbf{X}_{i-1}^q . When the activation function is monotone, the computation of \mathbf{X}_i^q can all be performed by operations on fixed-point numbers, even when the bias parameters \mathbf{b}_i are floating point numbers. The method is detailed in Appendix A.

2.4 Training Quantized Neural Networks by Straight-Through Estimator

Having quantization steps in computation prevents direct training of QNNs with the BP algorithm, as mathematically any quantization function will have zero derivatives. To remedy this, Courbariaux *et al.* [57] proposed to use STE to assign non-zero gradients for quantization functions. As the discrete parameters cannot be used to accumulate the high precision gradients, they kept two copies of parameters, one consisting of quantized values \mathbf{W}_q and the other consisting of real values \mathbf{W} . The real value version \mathbf{W} is used for accumulation, while \mathbf{W}_q is used for computation in forward and backward passes. We will refer to \mathbf{W}_q as quantized parameters, or simply parameters, of QNNs, and reserve \mathbf{W} for the “floating point copy” in the rest of this paper.

As STE introduces approximation noises into computations of gradients, we would like to limit it to places where necessary. It can be observed that the only function in Formula 1 that has the zero gradients is the rounding function. Hence we construct its STE version, *round-to-zero*_{ste}, as follows:

$$\begin{aligned} \text{Forward: } & \tilde{\mathbf{W}} \leftarrow \text{round-to-zero}(\mathbf{W}) \\ \text{Backward: } & \frac{\partial C}{\partial \mathbf{W}} \leftarrow \frac{\partial C}{\partial \tilde{\mathbf{W}}}, \end{aligned}$$

where $\tilde{\mathbf{W}}$ is the rounded value and C is the objective function used in training of the neural network.

Functions using *round-to-zero* function, like the k -bit uniform quantization function $quant_k$, can be transformed into the STE version by replacing *round-to-zero* with *round-to-zero*_{ste}.

A QNN can then use $quant_{ste}$ to include quantization in its computation graph. For completeness, we provide the inference and training algorithm of an L -layer QNN as Algorithm 3 in Appendix B.

3 Balanced Quantization for Neural Network Parameters

In this section, we focus on more effective quantization of parameters of QNNs to improve their prediction accuracies. We propose the Balanced Quantization method, which induces the quantized parameters to have balanced distributions. The method divides parameters by percentiles into bins

containing the same number of entries before the quantization. We also propose to use approximate thresholds in the algorithm to reduce computation overhead during training.

3.1 Effective Bitwidth and Prediction Accuracy of QNN

Using QNNs can reduce computation resource requirements considerably. However, QNNs usually have lower prediction accuracies than their floating point counterparts, especially when bitwidths goes below 4-bit [51, 52, 53].

We investigate this inefficiency of using low bitwidth parameters by inspecting the parameters of QNNs before and after the quantization. On many such models, we observe that the parameters before the quantization follow bell-shaped distributions, just as other DNNs [58, 59]. Moreover, it is not rare to observe outliers. Consequently, the quantized values after uniform quantization will often follow imbalanced distributions between possible values. An illustrative example is given in Fig. 1 and Fig. 2, where histograms of weight parameters, before and after the quantization, of a layer in a quantized ResNet model are shown. The quantized weights are 2-bit.

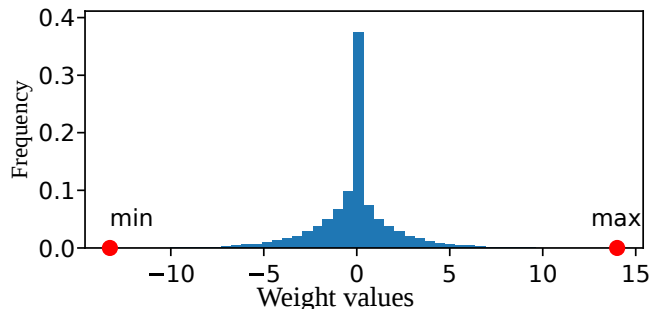


Figure 1: Floating point copy of weights in a QNN after 60 epochs of training. The weight values follow a bell-shaped distribution, and the minimum and maximum values differ a lot from the other values.

A QNN with parameters following imbalanced distributions may be sub-optimal. For example, the 2-bit weight model in Fig. 2 fails to exploit available value range, and may be well approximated with a 1-bit weight model.

Hence the “real” bitwidth of a QNN may be well below its specified bitwidth. To quantitatively measure the “effective” bitwidth, we propose to use the mean of entropy of parameters of each layer in a QNN as an

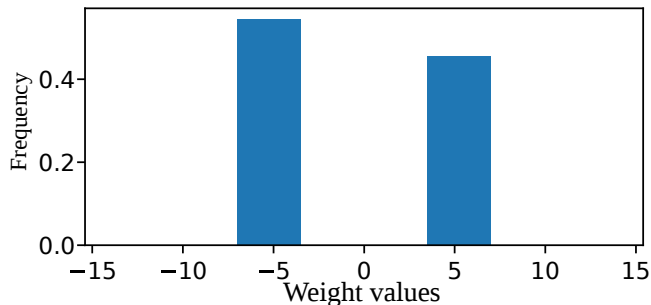


Figure 2: Results of imbalanced quantization (no equalization). After uniform quantization of weight values to 2-bit numbers, the quantized values concentrate on the central two out of four possible quantized values.

indicator as follows.

Definition 2

$$\begin{aligned} \text{effective-bitwidth}(\mathbf{x}) &\stackrel{\text{def}}{=} \text{entropy}(\mathbf{P}(\mathbf{x})) \\ &= \text{bitwidth} \times \frac{\text{entropy}(\mathbf{P}(\mathbf{x}))}{\text{entropy}(\text{UniformDistribution})}, \end{aligned}$$

where *entropy* is defined with base-2 logarithm, and $\mathbf{P}(\mathbf{x})$ refers to the distribution of \mathbf{x} .

The definition is in agreement with the following intuitions.

1. If the quantized values are concentrated in a few bins like in Fig. 2, indicating poor utilization of the available bitwidth, the Effective Bitwidth will be low, just as expected.
2. When the order of the bars standing for quantized values in histogram is permuted, which does not increase bitwidth utilization, the Effective Bitwidth will not change.
3. If \mathbf{x} is drawn from a discrete uniform distribution with 2^B possible values, then $\text{effective-bitwidth}(\mathbf{x}) = B$ as desired.

Based on this definition of Effective Bitwidth, we make the following conjecture that will be empirically validated in Subsection 4.2.1:

Conjecture 1 *Assume other factors affecting prediction accuracies, like learning rate schedules and model architectures, are kept the same. The*

prediction accuracy of a converged QNN model is positively correlated with its Effective Bitwidth.

Motivated by the conjecture, we propose a novel quantization algorithm that can enforce the balanced distribution of quantized parameters, which maximizes entropy of the converged model and consequently its Effective Bitwidth.

3.2 Balanced Quantization Algorithm

3.2.1 Outline

In this subsection we propose an algorithm to induce parameters of QNNs to have more balanced distributions, and consequently larger Effective Bitwidths.

The first step is histogram equalization, which can be implemented as a piecewise linear transform. The second step performs quantization, and then matches the value range with that of the input by an affine transformation.

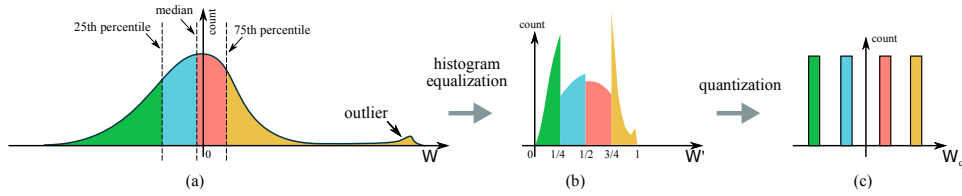


Figure 3: The schematic description of the Balanced Quantization algorithm in presence of outliers, with the case of $k = 2$ as an example. The histogram of the weight values is first equalized by piecewise linear transform and then mapped to a symmetric distribution. The subfigures are (a) the histogram of floating-point weight values, (b) the histogram-equalized weight values, and (c) the quantized weight values.

Fig. 3 gives a schematic diagram of the Balanced Quantization method. The method starts by partitioning numbers into bins containing the same number of entries. Each partition is then mapped to an evenly-divided interval in the closed interval $[0, 1]$. Finally, the quantization step maps intervals into discrete values and transforms the value range to be approximately the same as input. There will be exactly the same number of quantized values assigned to possible choices when percentiles are used as thresholds.

Algorithm 1 gives a more rigorous description of the whole process.

Algorithm 1: k -bit Balanced Quantization Algorithm of Matrix \mathbf{W}

Require: \mathbf{W} is a real matrix

Ensure : \mathbf{W}_q is quantized weights.

- 1 $scale \leftarrow \max(|\mathbf{W}|)$
{Histogram Equalization}
{The equalized values \mathbf{W}_e are in closed interval $[0, 1].$ }
 - 2 $\mathbf{W}_e \leftarrow equalize_k(\mathbf{W})$
{Quantization and restoring value range}
{ \mathbf{W}_f are fixed point numbers among 2^k discrete values
 $-\frac{1}{2}, -\frac{1}{2} + \frac{1}{2^{k-1}}, -\frac{1}{2} + \frac{2}{2^{k-1}} \cdots, \frac{1}{2}.$ }
 - 3 $\mathbf{W}_f \leftarrow \frac{1}{2^{k-1}} round\text{-to-zero}(2^k \mathbf{W}_e - \frac{1}{2}) - \frac{1}{2}$
{Values of \mathbf{W}_q are scaled fixed point numbers in closed
interval $[-\max(|\mathbf{W}|), \max(|\mathbf{W}|)].$ }
 - 4 $\mathbf{W}_q \leftarrow 2 \times scale \times \mathbf{W}_f$
-

3.2.2 Histogram Equalization by Piecewise Linear Transform

In this subsection, we detail the histogram equalization step, which we adapt from image processing literature [60].

Assume we are quantizing to k -bits values and $N = 2^k$. The input value range is divided to N intervals, including $N - 1$ number of half open intervals $[t_i, t_{i+1})$ and a closed interval $[t_{N-1}, t_N]$. To simplify notation, we denote the i -th interval as I_i . Thresholds $\{t_i\}_{i=0}^N$ are determined by the algorithm of histogram equalization. When exact equalization is desired, we let thresholds t_i be the $\frac{100i}{N}$ -th percentiles of the original distribution. The formula of equalized values x_e is as follows.

Definition 3 (Histogram equalization)

$$x_e = equalize_k(x) \stackrel{\text{def}}{=} a_i x + b_i$$

if $x \in I_i, \quad 0 \leq i \leq N - 1, i \in \mathbb{Z}.$

As $equalize_k$ maps I_i to evenly spaced segments J_i of target interval $[0, 1]$, parameters of the affine transformations, a_i and b_i , can be determined

from the following constraints:

$$\begin{aligned} a_i t_i + b_i &= \frac{i}{N}, \\ a_i t_{i+1} + b_i &= \frac{i+1}{N}, \quad 0 \leq i \leq N-1, \end{aligned}$$

where $\frac{i}{N}$ and $\frac{i+1}{N}$ are the two endpoints of J_i .

Let C be the objective function of the training, the back-propagation formula for $equalize_k$ is straightforward:

$$\begin{aligned} \frac{1}{a_i} \frac{\partial C}{\partial x} &= \frac{\partial C}{\partial x_e} \\ \text{if } x_e &\in J_i, \quad 0 \leq i \leq N-1, i \in \mathbb{Z}. \end{aligned}$$

3.2.3 Rounding and Restoring Value Range

After the histogram equalization step, the values \mathbf{W}_e are still floating point values, and need be converted to discrete values. The conversion can be done by the construction of fixed point version $\mathbf{W}_f = \frac{1}{2^k-1} \text{round-to-zero}(2^k \mathbf{W}_e - \frac{1}{2}) - \frac{1}{2}$. Note the mapping between W_e and W_f is different from Q_k . For example, it maps the interval of $[0, \frac{1}{2^k}]$ to 0, while Q_k maps $[0, \frac{1}{2(2^k-1)}]$ to 0.

Finally, \mathbf{W}_f , which has value range $[-\frac{1}{2}, \frac{1}{2}]$, can be scaled by $2 \max(|\mathbf{W}|)$ to match the original value range.

3.3 Approximation of Median and Efficient Implementation

The histogram equalization defined by the piecewise linear transform in Definition 3 has well-defined gradients and can be readily integrated into the training process of QNNs. However, a naive implementation using percentiles as thresholds would require sorting of weight values during each forward operation in BP, which may slow down the training process of QNNs as sorting is less efficient on modern hardware than matrix multiplications. In this subsection, we discuss an approximate equalization that allows efficient implementation. We first propose a recursive implementation of histogram equalization that only requires computing medians. Noting that medians can be well approximated by means, we construct Algorithm 2 that can perform approximate histogram equalization without doing sorting.

Algorithm 2: Histogram Equalization of Matrix \mathbf{W} by Recursive Partitioning

```

1 Function HistogramEqualize( $\mathbf{W}$ ,  $\mathbf{M}$ ,  $level$ ) Data:
    $\mathbf{W}$  is a real-valued matrix;
    $\mathbf{M}$  is a mask matrix with values in  $\{0,1\}$  and has the same shape
   as  $\mathbf{W}$ ; It is used to note the “working set” of  $\mathbf{W}$ .
    $level$  is an auxiliary variable recording recursion level.
Result: A matrix of the same shape as  $\mathbf{W}$  with value range  $[0, 1]$ 
   { $S_{\mathbf{W}}$  is the subset of the elements of  $\mathbf{W}$  with positive
   masks.}
2  $S_{\mathbf{W}} \leftarrow \{w_{i,j} | \forall w_{i,j} \in \mathbf{W}; m_{i,j} > 0\}$ 
3 if  $level = 0$  then
   | {Affine transform  $\mathbf{W}$  to the value range of  $[0,1]$ .}
   | { $*$  is element-wise (Hadamard) multiplication.}
4   return  $\frac{\mathbf{W} - \min(S_{\mathbf{W}})}{\max(S_{\mathbf{W}}) - \min(S_{\mathbf{W}})} \circ \mathbf{M}$ 
5 end

   {Construct two masks  $\mathbf{M}^l$  and  $\mathbf{M}^g$  using  $\text{mean}(S_{\mathbf{W}})$  as
   threshold.}
   { $\text{mean}(S_{\mathbf{W}})$  is used to replace  $\text{median}(S_{\mathbf{W}})$  so as to
   accelerate computation (see Section 3.3).}
6  $T \leftarrow \text{mean}(S_{\mathbf{W}}) = \frac{\sum(\mathbf{W} \circ \mathbf{M})}{\sum \mathbf{M}}$ 
7  $\mathbf{M}^l \leftarrow \mathbf{0}$ ,  $\mathbf{M}^g \leftarrow \mathbf{0}$ 
8 for  $w_{i,j} \in \mathbf{W}; m_{i,j} > 0$  do
9   | if  $w_{i,j} < T$  then
10  | |  $m_{i,j}^l \leftarrow 1$ 
11  | else
12  | |  $m_{i,j}^g \leftarrow 1$ 
13  | end
14 end
15  $\mathbf{W}^l \leftarrow \text{HistogramEqualize}(\mathbf{W}, \mathbf{M}^l, level-1)$ 
16  $\mathbf{W}^g \leftarrow \text{HistogramEqualize}(\mathbf{W}, \mathbf{M}^g, level-1)$ 
   {Value ranges of both  $\mathbf{W}^l$  and  $\mathbf{W}^g$  are  $[0,1]$ .}
   { $\frac{1}{2}$  is added to  $\frac{1}{2}\mathbf{W}^g$  to shift the value range to  $[\frac{1}{2},1]$ .}
17 return  $\frac{1}{2}\mathbf{W}^l + (\frac{1}{2}\mathbf{W}^g + \frac{1}{2}) \circ \mathbf{M}^g$ 

```

3.3.1 Recursive Partitioning

We first note that the 2^k evenly spaced percentiles required in histogram equalization can be computed from the recursive application of partitioning of numbers by medians. For example, when doing histogram equalization for the 2-bit quantization, we need to compute the 25-th, the 50-th and the 75-th percentiles as thresholds. However, the 50-th percentile is exactly the median, while the 25-th percentile (25% of values are below this number) is the median of those values that are below the median of the original distribution. Hence we can replace the computation of percentiles with recursive applications of partitioning by medians.

Moreover, we note that when a distribution has bounded variance σ , the mean μ approximates the median m as there is an inequality bounding the difference [61]:

$$|\mu - m| \leq \sigma.$$

Hence we may use means instead of medians in the recursive partitioning. The results of partitioning by different methods are shown in Fig. 4 and Fig. 5. It can be observed that partitioning by the median achieves perfect balance, and partitioning by the mean achieves approximate balance.

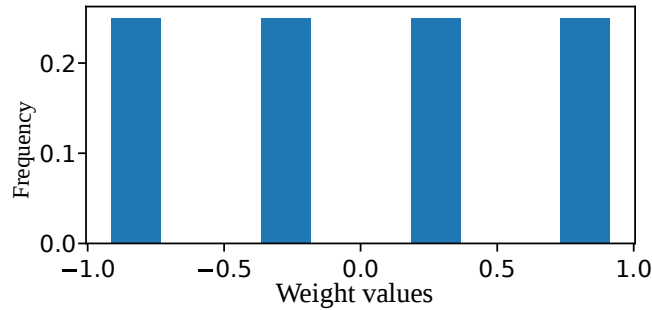


Figure 4: Balanced quantization with median (before matching value range)

3.3.2 Implementation

Based on the fact that the median approximates the mean, histogram equalization can be implemented as in Algorithm 2. An auxiliary mask matrix M , whose values are either 0 or 1, is introduced to help manipulate the

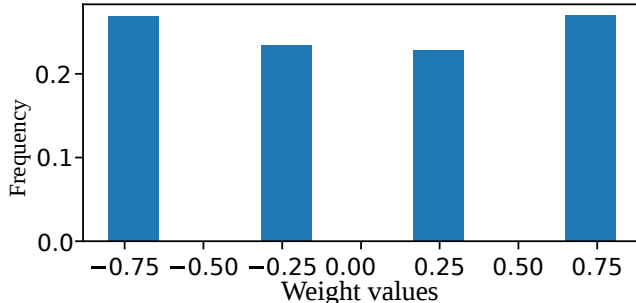


Figure 5: Balanced quantization with mean (before matching value range)

branching and selection operations. Note the mask \mathbf{M} , which is an argument of HistogramEqualize at the top of call chain, is initialized to be $\mathbf{1}$, a matrix with all values being 1.

When Algorithm 2 is used as the histogram equalization step in Algorithm 1, we can prove the following proposition (see Appendix C for proof):

Proposition 1 If during application of Algorithm 2 the following holds after Line 14:

$$\frac{1}{\gamma} \leq \frac{\sum \mathbf{M}^l}{\sum \mathbf{M}^g} \leq \gamma,$$

then the most frequent entry of the quantized values will appear at most γ^{2K} as often as that of the least frequent entry, when quantizing to K -bit numbers with Algorithm 1.

4 Experiments

In this section we empirically validate the effectiveness of the Balanced Quantization through experiments on quantized Convolutional Neural Networks and Recurrent Neural Networks.

In our implementations of QNNs, we convert parameters and input activations of all layers in the network to low bitwidth number, which is in line with the practice of Hubara *et al.* [51]. The CNN models used in this section are all equipped with Batch Normalization [62] to speed up convergence. Experiments are done on Linux machines with Intel Xeon CPUs and NVidia TitanX Graphic Processing Units.

4.1 Experiments on Convolutional Neural Networks

4.2 Datasets

For evaluation on CNNs, we conduct experiments on two datasets used for the image classification task.

The SVHN dataset [63] is a real-world digit recognition dataset consisting of photos of house numbers in Google Street View images. We consider the “cropped” format of the dataset: 32-by-32 colored images centered around a single character. We also include the “extra” part of labeled data in training.

The ImageNet dataset contains 1.2M images for training and 50K images for validation. Each image in the dataset is assigned a label in one of the 1000 categories. While testing, images are first resized such that the shortest edge is 256 pixels, and then the center 224-by-224 crops are fed into models. Following the conventions, we report results in two measures: single-crop top-1 error rate and top-5 error rate over ILSVRC12 validation sets [64]. For brevity, we will denote the top-1 and top-5 error rates as “top-1” and “top-5”, respectively.

4.2.1 Effective Bitwidths and Prediction Accuracies of Converged Models

In Fig. 6 and Fig. 7, we plot the prediction accuracies of several converged QNNs against their Effective Bitwidths as defined in Definition 2. The QNNs are trained on the SVHN dataset and have the same 7-layer CNN model architecture; hyper-parameters like learning rate schedule, numbers of epochs are kept the same, such that the differences between these models are only the specified bitwidths of parameters and the quantization methods. In this way, we can evaluate the impact of Effective Bitwidths on the prediction accuracies of converged models.

It can be observed from Fig.6 that in general, accuracy grows with the increase of Effective Bitwidth. However, the growth of the accuracy gradually slows down to the right half of the diagram, when the prediction accuracy of a quantized model approaches the upper bound set by floating point models.

4.2.2 Evaluation of Approximation of Median

In this subsection we validate the effectiveness of approximation of the median by the mean, as proposed in Subsection 3.3. As computing the median requires doing sorting of weight parameters of a layer, experiments on DNNs with many parameters will be very slow. Hence we perform experiments on

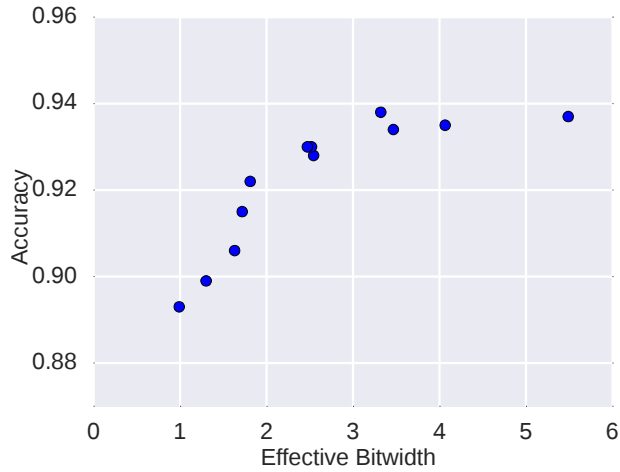


Figure 6: Relationship between Effective Bitwidths and prediction accuracies of several converged QNNs on the SVHN dataset. The models are produced by different specified bitwidths (ranging from 1-bit to 8-bit) and quantization methods (balanced or not), but all have the same architecture and training settings.

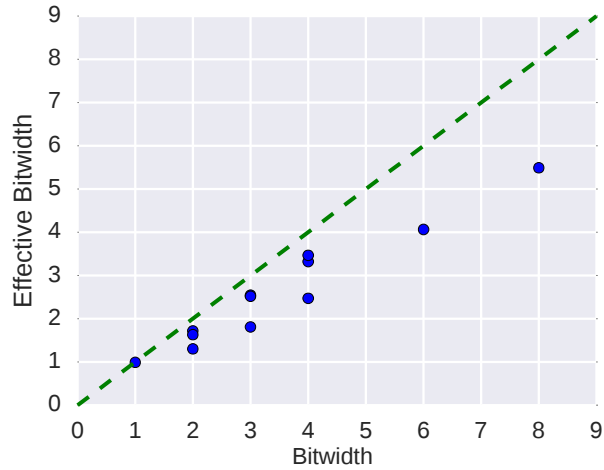


Figure 7: Relationship between Effective Bitwidth and specified Bitwidth. In general, Effective Bitwidths grow with Bitwidths. But Effective Bitwidths of most of the models are significantly less than its specified Bitwidth.

Table 1: Evaluation of using means instead of medians when performing Balanced Quantization, on GoogLeNet with 4-bit weights and 4-bit activations.

Thresholds	Top-1	Top-5	Effective Bitwidth
mean	32.3%	12.7%	3.99
median	33.8%	13.3%	4.00

Table 2: Comparison of performances of quantized AlexNet and ResNet.

Method	AlexNet			ResNet-18		
	Top-1	Top-5	Effective Bitwidth	Top-1	Top-5	Effective Bitwidth
FP	42.9%	20.6%	-	31.8%	12.5%	-
equalized FP weights	42.7%	20.9%	-	36.2%	15.3%	-
FP weight + 2-bit feature	43.5%	21.0%	-	38.9%	17.3%	-
imbalanced 2-bit (different settings)	46.4%	24.7%	1.89	46.6%	22.1%	0.99
imbalanced 2-bit	45.3%	22.3%	1.94	42.3%	19.2%	1.96
balanced 2-bit	44.3%	22.0%	1.99	40.6%	18.0%	1.99

FP stands for floating point. Results in rows prefixed with “imbalanced” are produced from direct applications of uniform quantization. Results in rows marked with “equalized FP weights” only perform equalization of weights on FP models. As the floating point values do not have well-defined effective bitwidths, we omit these entries by using the “-” symbols.

the GoogLeNet, which contains fewer than 7M parameters. From Table 1, it can be seen that replacing medians by means does not degrade prediction accuracies. In fact, the method using means as thresholds is even slightly better than the method using medians, both in terms of top-1 and top-5 error rates.

As replacing medians with means is empirically found to be viable, we will use means as thresholds in experiments in the rest of Section 4.

4.2.3 Balanced Quantization of AlexNet, ResNet-18 and GoogLeNet

The experiment results on AlexNet and ResNet-18 are summarized in Table 2. The results marked with “different settings” come from models trained with a different learning rate schedule and clipping of weights. It can be seen that results of the Balanced Quantization method consistently outperform those of the uniform quantization methods (hereafter denoted as Imbalanced Quantization) defined in Definition 1. In particular, the top-5 error rate of the Balanced Quantized 2-bit AlexNet is within 2 percentages of that of the floating point version, making the quantized network a good candidate to replace the floating point version in practice. As the model size can be reduced to $\frac{1}{16}$ of the original and computations can be performed by 2-bit numbers, the savings in resource requirements will be significant.

However, the improvements of accuracies due to Balanced Quantization may not be large, as accuracies of models quantized without balance are already close to the upper bounds set by models with floating point weights and 2-bit features.

Table 3: Comparison of classification error rates with state-of-the-arts on quantized GoogLeNet model with 4-bit weights and 4-bit activations.

Method	Top-1	Top-5
Our float32	28.5%	10.1%
QNN 4-bit [51]	33.5%	16.6%
Ristretto 8-bit [65]	33.4%	-
Our 4-bit	32.3%	12.7%

Table 3 compares QNNs quantized with our method with state-of-the-arts. It can be seen that our method consistently outperforms the others. In particular, our method reduces the top-5 accuracy degradation, which is the difference in accuracy between a QNN and a floating point version, from 6.5 percentages to 2.6 percentages.

Table 4: Performance of Quantized RNNs on PTB datasets.

Model	w-bits	a-bits	PPW		Effective Bitwidth	
			balanced	imbalanced	balanced	imbalanced
GRU	2	2	142	165	1.98	1.56
GRU	4	4	116	120	3.86	3.26
GRU (tanh(\mathbf{W}))	FP	FP	-	118	-	-
GRU	FP	FP	-	100	-	-
LSTM	2	2	126	164	1.96	1.00
LSTM	2	3	123	155	1.95	1.00
LSTM [51]	2	3		220		
LSTM	4	4	114	127	3.89	1.80
LSTM [51]	4	4		100		
LSTM (tanh(\mathbf{W}))	FP	FP	-	122	-	-
LSTM	FP	FP	-	106	-	-
LSTM [51]	FP	FP		97		

FP stands for 32-bit floating point. Results marked with tanh(\mathbf{W}) are of models that have their weights clipped by tanh before passing to quantization. The best results for each bitwidth setting are marked in bold.

4.2.4 Break-down of Accuracy Degradation with Balanced Quantization

Overall, the change in accuracy due to Balanced Quantization will be made up of two parts:

$$\Delta\text{Accuracy}_{\text{total}} = \Delta\text{Accuracy}_{\text{eq}} + \Delta\text{Accuracy}_{\text{quant}},$$

where $\Delta\text{Accuracy}_{\text{eq}}$ stands for the change in accuracy due to equalization and $\Delta\text{Accuracy}_{\text{quant}}$ is that of quantization.

The histogram equalization effectively imposes an additional constraint on the neural network parameters. As the constraint limits the optimization space of parameters, it will likely introduce additional errors into predictions of neural networks.

Nevertheless, through the experiments in Table 2, we have observed that the reduction in $\Delta\text{Accuracy}_{\text{quant}}$ outweighs the inclusion of additional term $\Delta\text{Accuracy}_{\text{eq}}$. We leave it as future work to investigate the cause and further reduction of $\Delta\text{Accuracy}_{\text{eq}}$.

4.3 Experiments on Recurrent Neural Networks

In this subsection we evaluate the effect of Balanced Quantization on a few Recurrent Neural Networks. We take language modeling task as an example, and use the Penn Treebank dataset [66], which contains 10K unique words.

For fair comparison, in the following experiments, all of our models use one hidden layer with 300 hidden units, which is the same with [51]. A word embedding layer is used at the input side of the network whose weights are trained from scratch. The performance is measured in perplexity per word (PPW) metric.

During experiments we find the magnitudes of weights often grow rapidly with training when using small bitwidth, and may result in divergence. This can be alleviated by adding tanh to constrain the value ranges [53] and adding weight decays for regularization. However, we find using tanh to clip parameters will degrade prediction accuracy of floating point Neural Network. Further investigation of this drop of accuracy is out of the scope of this paper, and will be left as future work.

Experiment results are reported in Table 4. Our result is in agreement with [51] in finding that using 4-bit weights and activations can achieve comparable performance as floating point counterparts. However, we report higher accuracy than [51] when using less bits, such as 2-bit weight and activations. In particular, our 2-bit weights and 3-bit activations LSTM achieve 155 PPW for imbalanced quantization and 123 PPW for balanced quantization, both of which outperform the counterparts in [51] by large margins, despite that our floating point models are worse than those of [51].

5 Conclusions

In this paper, we have introduced the method of Balanced Quantization, which enforces the quantized values to have balanced distributions through the use of histogram equalization. Our method breaks away from traditional quantization methods in that it emphasizes on shaping distributions of quantized values. When incorporated into the training process of Quantized Neural Networks, our method can improve the prediction accuracies of converged models. We have also introduced Effective Bitwidth, which measures the utilization of bitwidth in QNNs, that can help identify models that can benefit more from the Balanced Quantization method.

To reduce the computation overhead introduced by the need to compute percentiles when performing Balanced Quantization, we also propose to use recursive application of mean as approximations of percentiles (see

Subsection 3.3). We have also applied the Balanced Quantization method to several popular Neural Network architectures like AlexNet, GoogLeNet and ResNet, and found that our method outperforms the state-of-the-arts of QNN, in terms of prediction accuracy (see Subsection 4.2.3). Experiments on LSTM and GRU are also encouraging (see Subsection 4.3).

As future work, it would be interesting to use the histogram transformation technique to induce distributions that have other benefits, like a high ratio of zeros in quantized values. It would also be interesting to investigate whether inducing activations of neural networks to have balanced distributions could improve the prediction accuracies of QNNs.

Appendix

A Eliminating All Floating Point Operations During Inference

Recall that the i -th layer of a QNN is like:

$$\begin{aligned}\mathbf{X}_i^q &= Q_A(\sigma_i(\mathbf{W}_i^q \mathbf{X}_{i-1}^q + \mathbf{b}_i)) \\ \mathbf{W}_i^q &= Q_W(\mathbf{W}_i),\end{aligned}$$

where σ_i is the activation function, and Q_A and Q_W are quantization functions.

Below we assume the following conditions:

1. \mathbf{W}_i^q can be represented as fixed point numbers scaled by a floating point scalar α , i.e. $\mathbf{W}_i^q = \alpha \mathbf{W}_f$, where \mathbf{W}_f is fixed point numbers.
2. \mathbf{X}_{i-1}^q contains only fixed point numbers
3. σ_i is a monotone function

We next show that under these assumptions, the computation of \mathbf{X}_i^q can be done by operations between fixed point numbers.

First, by replacing the variables we have:

$$\mathbf{X}_i^q = Q_A(\sigma_i(\alpha \mathbf{W}_f \mathbf{X}_{i-1}^q + \mathbf{b}_i)).$$

As Q_A is a uniform quantization function, it can be computed by comparing values of $\sigma_i(\alpha \mathbf{W}_f \mathbf{X}_{i-1}^q + \mathbf{b}_i)$ with a sequence of thresholds h_1, h_2, \dots, h_n .

As σ_i is monotone, and w.l.o.g. assume $\alpha > 0$, the comparison can equivalently be done between $\mathbf{W}_f \mathbf{X}_{i-1}^q$ and

$$\frac{1}{\alpha}(\sigma_i^{-1}(h_1) - b), \frac{1}{\alpha}(\sigma_i^{-1}(h_2) - b), \dots, \frac{1}{\alpha}(\sigma_i^{-1}(h_n) - b).$$

As \mathbf{W}_f and \mathbf{X}_{i-1}^q are fixed point numbers, their product is necessarily made of fixed point numbers, hence there exists a sufficiently large integer K such that $2^K \mathbf{W}_f \mathbf{X}_{i-1}^q$ are integers. The comparison required for computing Q_A can be done by comparing $2^K \mathbf{W}_f \mathbf{X}_{i-1}^q$ with integers

$$\lfloor \frac{2^K}{\alpha}(\sigma_i^{-1}(h_1) - b) \rfloor, \lfloor \frac{2^K}{\alpha}(\sigma_i^{-1}(h_2) - b) \rfloor, \dots, \\ \lfloor \frac{2^K}{\alpha}(\sigma_i^{-1}(h_n) - b) \rfloor.$$

Hence the computation of \mathbf{X}_i^q can be done by the comparison between fixed point numbers $\mathbf{W}_f \mathbf{X}_{i-1}^q$ with the following thresholds that can be pre-computed and stored (hence eliminating need for floating point operations during inference):

$$2^{-K} \lfloor \frac{2^K}{\alpha}(\sigma_i^{-1}(h_1) - b) \rfloor, 2^{-K} \lfloor \frac{2^K}{\alpha}(\sigma_i^{-1}(h_2) - b) \rfloor, \\ \dots, 2^{-K} \lfloor \frac{2^K}{\alpha}(\sigma_i^{-1}(h_n) - b) \rfloor$$

B Training Algorithm of QNN

For completeness we outline the training algorithm of QNNs in Algorithm 3. Weights, activations and gradients are quantized by quantization functions Q_W , Q_A and Q_G , that are applied to weights, activations and gradients respectively. C stands for the cost function of the neural network. *backward_input* and *backward_weight* are functions derived from chain rule for computing gradients with respect to inputs and weights, respectively. The Update function is determined by the learning rule used. The algorithm extends Algorithm 1 in Hubara *et al.* [51] to include the quantization of gradients, and the multi-bit quantization.

Algorithm 3: Training a L -layer CNN with W -bit weights and A -bit activations using G -bit gradients.

Require: a minibatch of inputs and labels $(\mathbf{X}_0, \mathbf{Y})$, previous weights \mathbf{W} , learning rate η

Ensure : updated weights \mathbf{W}^{t+1}

{1. Computing the parameter gradients:}

{1.1 Forward Propagation:}

1 **for** $i = 1 \rightarrow L$ **do**

2 $\mathbf{W}_i^q \leftarrow Q_W(\mathbf{W}_i)$

3 $\tilde{\mathbf{X}}_i \leftarrow \mathbf{X}_{i-1}^q \mathbf{W}_i^q + \mathbf{b}_i$

4 $\mathbf{X}_i \leftarrow \sigma(\tilde{\mathbf{X}}_i)$

5 **if** $k < L$ **then**

6 $\mathbf{X}_i^q \leftarrow Q_A(\mathbf{X}_i)$

7 **end**

8 Optionally apply pooling

9 **end**

{1.2 Backward propagation:}

10 Compute $g_L = \frac{\partial C}{\partial \mathbf{X}_L}$ knowing \mathbf{X}_L and label \mathbf{Y} .

11 **for** $i = L \rightarrow 1$ **do**

12 Back-propagate g_i through activation function σ

13 $g_i^q \leftarrow Q_G(g_i)$

14 $g_{i-1} \leftarrow \text{backward_input}(g_i^q, \mathbf{W}_i^q)$

15 $g_{\mathbf{W}_i} \leftarrow \text{backward_weight}(g_i^q, \mathbf{X}_{i-1}^q)$

16 Back-propagate gradients through pooling layer if there is one

17 **end**

{2. Accumulating the parameters gradients:}

18 **for** $k = 1 \rightarrow L$ **do**

19 $g_i = g_i^q \frac{\partial \mathbf{W}_i^q}{\partial \mathbf{W}_i}$

20 $\mathbf{W}_i^{t+1} \leftarrow \text{Update}(\mathbf{W}_i, g_i, \eta)$

21 **end**

C Proof of Proposition 1

Proof The step after histogram equalization in Algorithm 1 maps the following half open (close) intervals into quantization values:

$$[0, \frac{1}{2^K}), [\frac{1}{2^K}, \frac{2}{2^K}), \dots, [\frac{2^K - 1}{2^K}, 1].$$

Hence it is sufficient to prove the counting statements for these intervals after application of Algorithm 2.

First of all, as each call of HistogramEqualize either produces two recursive calls or terminates depending on *level* variable, the call relation of any invocation of HistogramEqualize will form a balanced binary tree. For clarity, we note as M_k^l, M_k^g the corresponding M^l, M^g used for a depth k node of the binary tree.

By the assumption of M^l, M^g , we have $\frac{1}{\gamma} \leq \frac{\sum M_k^l}{\sum M_k^g} \leq \gamma$. At the leaf nodes, the application will be at most K number of times, hence the number of entries in leaf nodes will be different by at most γ^{2k} number of times.

What remains to be proved is that no two leaf nodes produce the same quantized value. We create an auxiliary variable $D_n^k \in \{0, 1\}$ to record whether a depth k node is on the right branch of their depth $k - 1$ parent. We can prove that node n will map values to the interval $\sum_k D_n^k 2^{k-1}$, by observing that at Line 17 of Algorithm 2, $\frac{1}{2}$ will only be added if the right branch of the call tree is taken.

As $\sum_k D_n^k 2^{k-1}$ is unique for all nodes, we complete the proof. \square

D Quantization of GRU

We first investigate the quantization of GRU as it is structurally simpler. The basic structure of GRU cell may be described as follows:

$$\begin{aligned} z_t &= \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \\ r_t &= \sigma(\mathbf{W}_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \\ \widetilde{\mathbf{h}}_t &= \tanh(\mathbf{W} \cdot [r_t \circ \mathbf{h}_{t-1}, \mathbf{x}_t]) \\ \mathbf{h}_t &= (\mathbf{1} - z_t) \circ \mathbf{h}_{t-1} + z_t \circ \widetilde{\mathbf{h}}_t, \end{aligned}$$

where $\mathbf{1}$ is a vector with all entries being 1, σ stands for the sigmoid function, “ \cdot ” stands for the dot product, $[\mathbf{x}, \mathbf{y}]$ stands for the concatenation of two vectors \mathbf{x} and \mathbf{y} , and \circ stands for the Hadamard product.

Recall that to benefit from the speed advantage of bit convolution kernels, we need to multiply the two matrix inputs in low bit forms, such that the dot product can be calculated by bitwise operation. For plain feed forward neural networks, as the convolutions take up most of computation time, we can get decent acceleration by the quantization of inputs of convolutions and their weights. But when it comes to more complex structures like GRU, we need to check the bitwidth of each interlink.

Except for matrix multiplications needed to compute \mathbf{z}_t , \mathbf{r}_t and $\widetilde{\mathbf{h}}_t$, the gate structure of $\widetilde{\mathbf{h}}_t$ and \mathbf{h}_t brings in the need for element-wise multiplication. As the outputs of the sigmoid function may have higher bitwidths, the element-wise multiplication may need be done between floating point numbers (or in higher bitwidth format). As $\widetilde{\mathbf{h}}_t$ and \mathbf{h}_t are also the inputs to computations at the next timestamp, and noting that a quantized value multiplied by a quantized value will have a larger bitwidth, we need to insert additional quantization steps after element-wise multiplications.

Another problem with the quantization of GRU structure is that the value ranges of gates are different. The range of \tanh is $[-1, 1]$, which is different from the value range $[0, 1]$ of \mathbf{z}_t and \mathbf{r}_t . If we want to preserve the original activation functions, we will have the following quantization scheme:

$$\begin{aligned} \mathbf{z}_t &= \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \\ \mathbf{r}_t &= \sigma(\mathbf{W}_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \\ \widetilde{\mathbf{h}}_t &= \tanh(\mathbf{W} \cdot [2Q_k(\frac{1}{2}(\mathbf{r}_t \circ \mathbf{h}_{t-1}) + \frac{1}{2}) - 1, \mathbf{x}_t]) \\ \mathbf{h}_t &= 2Q_k(\frac{1}{2}((\mathbf{1} - \mathbf{z}_t) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \widetilde{\mathbf{h}}_t) + \frac{1}{2}) - \mathbf{1}, \end{aligned}$$

where we assume the weights W_z, W_r, W have already been quantized to the closed interval $[-1, 1]$, and input \mathbf{x}_t have already been quantized to $[-1, 1]$.

However, we note that the quantization function already has an affine transform to shift the value range. To simplify the implementation, we replace the activation functions of $\widetilde{\mathbf{h}}_t$ to be the sigmoid function, so that $(\mathbf{1} - \mathbf{z}_t) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \widetilde{\mathbf{h}}_t \in [0, 1]$.

Summarizing the above considerations, the quantized version of GRU could be written as

$$\begin{aligned} \mathbf{z}_t &= \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \\ \mathbf{r}_t &= \sigma(\mathbf{W}_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t]) \\ \widetilde{\mathbf{h}}_t &= \sigma(\mathbf{W} \cdot [Q_k(\mathbf{r}_t \circ \mathbf{h}_{t-1}), \mathbf{x}_t]) \\ \mathbf{h}_t &= Q_k((\mathbf{1} - \mathbf{z}_t) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \widetilde{\mathbf{h}}_t), \end{aligned}$$

where we assume the weights $\mathbf{W}_z, \mathbf{W}_r, \mathbf{W}$ have already been quantized to $[-1, 1]$, and input \mathbf{x}_t have already been quantized to $[0, 1]$.

E Quantization of LSTM

The structure of LSTM can be described as follows:

$$\begin{aligned}
 \mathbf{f}_t &= \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \\
 \mathbf{i}_t &= \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \\
 \widetilde{\mathbf{C}}_t &= \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \\
 \mathbf{C}_t &= \mathbf{f}_t \circ \mathbf{C}_{t-1} + \mathbf{i}_t \circ \widetilde{\mathbf{C}}_t \\
 \mathbf{o}_t &= \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \\
 \mathbf{h}_t &= \mathbf{o}_t \circ \tanh(\mathbf{C}_t)
 \end{aligned}$$

Different from GRU, \mathbf{C}_t cannot be easily quantized, since the value has not been bounded by activation functions like tanh. This difficulty comes from structure design and cannot be alleviated without introducing extra facility to clip value ranges. But it can be noted that the computations involving \mathbf{C}_t are all element-wise multiplications and additions, which may take much less time than computing matrix products. For this reason, we leave \mathbf{C}_t to be floating point numbers.

To simplify implementation, tanh activation for output may be changed to the sigmoid function.

Summarizing above changes, the formula for quantized LSTM can be:

$$\begin{aligned}
 \mathbf{f}_t &= \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \\
 \mathbf{i}_t &= \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \\
 \widetilde{\mathbf{C}}_t &= \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \\
 \mathbf{C}_t &= \mathbf{f}_t \circ \mathbf{C}_{t-1} + \mathbf{i}_t \circ \widetilde{\mathbf{C}}_t \\
 \mathbf{o}_t &= \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \\
 \mathbf{h}_t &= Q_k(\mathbf{o}_t \circ \sigma(\mathbf{C}_t)),
 \end{aligned}$$

where we assume the weights $\mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_C, \mathbf{W}_o$ have already been quantized to $[-1, 1]$, and input \mathbf{x}_t have already been quantized to $[0, 1]$.

References

- [1] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks. In *Proc. Advances in neural information processing systems*, Dec. 2012, pp. 1097–1105.
- [2] Zeiler M D, Fergus R. Visualizing and understanding convolutional networks. In *Proc. European Conference on Computer Vision*, Sep. 2014, pp. 818–833.
- [3] Girshick R, Donahue J, Darrell T, Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. IEEE conference on Computer Vision and Pattern Recognition*, Jun. 2014, pp. 580–587.
- [4] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2015, pp. 3431–3440.
- [5] Hinton G, Deng L, Yu D, Dahl G E, Mohamed A r, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath T N et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 2012, 29(6):82–97.
- [6] Graves A, Mohamed A, Hinton G E. Speech recognition with deep recurrent neural networks. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2013, pp. 6645–6649.
- [7] Mikolov T, Sutskever I, Chen K, Corrado G S, Dean J. Distributed representations of words and phrases and their compositionality. In *Proc. Advances in neural information processing systems*, Dec. 2013, pp. 3111–3119.
- [8] Sutskever I, Vinyals O, Le Q V. Sequence to sequence learning with neural networks. In *Proc. Advances in neural information processing systems*, Dec. 2014, pp. 3104–3112.
- [9] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [10] Mnih V, Kavukcuoglu K, Silver D, Rusu A A, Veness J, Bellemare M G, Graves A, Riedmiller M, Fidjeland A K, Ostrovski G et al.

- Human-level control through deep reinforcement learning. *Nature*, 2015, 518(7540):529–533.
- [11] Silver D, Huang A, Maddison C J, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016, 529(7587):484–489.
- [12] He K, Zhang X, Ren S, Sun J. Identity mappings in deep residual networks. In *Proc. the 14th European Conference Computer Vision (ECCV)*, Oct. 2016, pp. 630–645.
- [13] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. *CoRR*, 2014, abs/1409.1556.
- [14] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S E, Anguelov D, Erhan D, Vanhoucke V, Rabinovich A. Going deeper with convolutions. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2015, pp. 1–9.
- [15] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2016, pp. 770–778.
- [16] Galal S, Horowitz M. Energy-efficient floating-point unit design. *IEEE Trans. Computers*, 2011, 60(7):913–922.
- [17] Hochreiter S, Schmidhuber J. Long short-term memory. *Neural computation*, 1997, 9(8):1735–1780.
- [18] Chung J, Gülçehre Ç, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, 2014, abs/1412.3555.
- [19] Pham P H, Jelaca D, Farabet C, Martini B, LeCun Y, Culurciello E. Neuflow: Dataflow vision processing system-on-a-chip. In *Proc. IEEE the 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2012, pp. 1044–1047.
- [20] Chen T, Du Z, Sun N, Wang J, Wu C, Chen Y, Temam O. Diannao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proc. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2014, pp. 269–284.

- [21] Luo T, Liu S, Li L, Wang Y, Zhang S, Chen T, Xu Z, Temam O, Chen Y. Dadiannao: A neural network supercomputer. *IEEE Trans. Computers*, 2017, 66(1):73–88.
- [22] Denton E L, Zaremba W, Bruna J, LeCun Y, Fergus R. Exploiting linear structure within convolutional networks for efficient evaluation. In *Proc. Advances in Neural Information Processing Systems*, Dec. 2014, pp. 1269–1277.
- [23] Jaderberg M, Vedaldi A, Zisserman A. Speeding up convolutional neural networks with low rank expansions. In *Proc. British Machine Vision Conference (BMVC)*, Sep. 2014.
- [24] Tai C, Xiao T, Wang X, E W. Convolutional neural networks with low-rank regularization. *CoRR*, 2015, abs/1511.06067.
- [25] Zhou S, Wu J, Wu Y, Zhou X. Exploiting local structures with the kronecker layer in convolutional networks. *CoRR*, 2015, abs/1512.09194.
- [26] Novikov A, Podoprikin D, Osokin A, Vetrov D P. Tensorizing neural networks. In *Proc. Advances in Neural Information Processing Systems*, Dec. 2015, pp. 442–450.
- [27] Zhang X, Zou J, He K, Sun J. Accelerating very deep convolutional networks for classification and detection. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 2016, 38(10):1943–1955.
- [28] Anwar S, Hwang K, Sung W. Structured pruning of deep convolutional neural networks. *CoRR*, 2015, abs/1512.08571.
- [29] Han S, Pool J, Tran J, Dally W J. Learning both weights and connections for efficient neural network. In *Proc. Advances in Neural Information Processing Systems*, Dec. 2015, pp. 1135–1143.
- [30] Han S, Mao H, Dally W J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, 2015, abs/1510.00149.
- [31] Liu B, Wang M, Foroosh H, Tappen M F, Pensky M. Sparse convolutional neural networks. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 806–814.

- [32] Cheng Y, Yu F X, Feris R S, Kumar S, Choudhary A N, Chang S. An exploration of parameter redundancy in deep networks with circulant projections. In *Proc. IEEE International Conference on Computer Vision*, Dec. 2015, pp. 2857–2865.
- [33] Chen W, Wilson J T, Tyree S, Weinberger K Q, Chen Y. Compressing neural networks with the hashing trick. In *Proc. the 32nd International Conference on Machine Learning*, Jul. 2015, pp. 2285–2294.
- [34] Chen W, Wilson J T, Tyree S, Weinberger K Q, Chen Y. Compressing convolutional neural networks in the frequency domain. In *Proc. the 22nd International Conference on Knowledge Discovery and Data Mining*, Aug. 2016, pp. 1475–1484.
- [35] Anguita D, Carlino L, Ghio A, Ridella S. A fpga core generator for embedded classification systems. *Journal of Circuits, Systems, and Computers*, 2011, 20(02):263–282.
- [36] Vanhoucke V, Senior A, Mao M Z. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning Workshop, NIPS*, Dec. 2011.
- [37] Alvarez R, Prabhavalkar R, Bakhtin A. On the efficient representation and execution of deep acoustic models. In *Proc. the 17th Annual Conference of the International Speech Communication Association*, Sep. 2016, pp. 2746–2750.
- [38] Zen H, Agiomyrgiannakis Y, Egberts N, Henderson F, Szczepaniak P. Fast, compact, and high quality LSTM-RNN based statistical parametric speech synthesizers for mobile devices. In *Proc. the 17th Annual Conference of the International Speech Communication Association, San Francisco*, Sep. 2016, pp. 2273–2277.
- [39] Gong Y, Liu L, Yang M, Bourdev L D. Compressing deep convolutional networks using vector quantization. *CoRR*, 2014, abs/1412.6115.
- [40] Merolla P, Appuswamy R, Arthur J V, Esser S K, Modha D S. Deep neural networks are robust to weight binarization and other non-linear distortions. *CoRR*, 2016, abs/1606.01981.
- [41] Gupta S, Agrawal A, Gopalakrishnan K, Narayanan P. Deep learning with limited numerical precision. *arXiv preprint arXiv:1502.02551*, 2015.

- [42] Courbariaux M, Bengio Y. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, 2016, abs/1602.02830.
- [43] Wu J, Leng C, Wang Y, Hu Q, Cheng J. Quantized convolutional neural networks for mobile devices. *CoRR*, 2015, abs/1512.06473.
- [44] Kim M, Smaragdis P. Bitwise neural networks. *CoRR*, 2016, abs/1601.06071.
- [45] Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y. Binarized neural networks. In *Proc. Advances in Neural Information Processing Systems*, Dec. 2016, pp. 4107–4115.
- [46] Rastegari M, Ordonez V, Redmon J, Farhadi A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proc. the 14th European Conference Computer Vision*, Oct. 2016, pp. 525–542.
- [47] Hinton G, Srivastava N, Swersky K. Neural networks for machine learning. *Coursera, video lectures*, 2012, 264.
- [48] Bengio Y, Léonard N, Courville A C. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, 2013, abs/1308.3432.
- [49] Hwang K, Sung W. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *Proc. IEEE Workshop on Signal Processing Systems*, Oct. 2014, pp. 174–179.
- [50] Shin S, Hwang K, Sung W. Fixed-point performance analysis of recurrent neural networks. In *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mar. 2016, pp. 976–980.
- [51] Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, 2016, abs/1609.07061.
- [52] Miyashita D, Lee E H, Murmann B. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.
- [53] Zhou S, Wu Y, Ni Z, Zhou X, Wen H, Zou Y. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *CoRR*, 2016, abs/1606.06160.

- [54] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado G S, Davis A, Dean J, Devin M et al. Tensorflow: Large-scale machine learning on heterogeneous systems. *Software available from tensorflow.org*, 2015.
- [55] Andri R, Cavigelli L, Rossi D, Benini L. Yodann: An ultra-low power convolutional neural network accelerator based on binary weights. In *Proc. IEEE Computer Society Annual Symposium on VLSI*, Jul. 2016, pp. 236–241.
- [56] Lee M, Hwang K, Park J, Choi S, Shin S, Sung W. Fpga-based low-power speech recognition with recurrent neural networks. In *Proc. IEEE International Workshop on Signal Processing Systems*, Oct. 2016, pp. 230–235.
- [57] Courbariaux M, Bengio Y, David J. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proc. Advances in Neural Information Processing Systems*, Dec. 2015, pp. 3123–3131.
- [58] Saxe A M, Koh P W, Chen Z, Bhand M, Suresh B, Ng A Y. On random weights and unsupervised feature learning. In *Proc. the 28th International Conference on Machine Learning*, Jun. 2011, pp. 1089–1096.
- [59] Giryes R, Sapiro G, Bronstein A M. Deep neural networks with random gaussian weights: A universal classification strategy? *IEEE Transactions on Signal Processing*, 2015, 64(13):3444–3457.
- [60] Heckbert P S. Color image quantization for frame buffer display. In *Proc. the 9th Annual Conference on Computer Graphics and Interactive Techniques*, Jul. 1982, pp. 297–307.
- [61] Mallows C. Another comment on o’cinneide. *The American Statistician*, 1991, 45(3):257.
- [62] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [63] Netzer Y, Wang T, Coates A, Bissacco A, Wu B, Ng A Y. Reading digits in natural images with unsupervised feature learning. In *Proc. Workshop on deep learning and unsupervised feature learning, NIPS*, volume 2011, 2011.

- [64] Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein M et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015, 115(3):211–252.
- [65] Gysel P, Motamedi M, Ghiasi S. Hardware-oriented approximation of convolutional neural networks. *CoRR*, 2016, abs/1604.03168.
- [66] Taylor A, Marcus M, Santorini B. *The Penn Treebank: An Overview*, pp. 5–22. Springer Netherlands, Dordrecht, 2003.