Quantization Schemes For Training Neural Networks

Cristian Ivan

Abstract

This work investigates the possibility of directly training quantized neural networks. First, a weight quantization scheme is proposed which shows that little to no performance loss is incurred when quantizing weights down to 4 bits. Second, it investigates the impact of a similar quantization scheme on the activations of neurons. Experiments performed with a simple fully connected network on the MNIST dataset as well as with a residual neural network on the CIFAR-10 dataset show no significant loss when training with 4 bit weights and activations.

1 Method

For an arbitrary feed-forward network with L hidden layers where each layer l has n_l units, the general expression for an output of node j in layer l is given by:

$$h_j^l = \sigma\left(\sum_{i=0}^{n_{l-1}} w_{ij}^l \cdot h_i^{l-1}\right) \tag{1}$$

where $\sigma(x)$ is the non-linear activation function, h_i^{l-1} is the output of a node from the previous layer and w_{ij}^l are the incoming weights for the current node. Note that we omit the biases for brevity.

Quantizing the weights implies introducing discontinuities in the distribution of the possible values taken by the weights. We would like to use a function which maps all weights within a certain interval Δw into a single value. Simple functions like $round(x)$, $ceil(x)$, $floor(x)$ already satisfy the given requirement, although they can be converted between each other by just adding or subtracting a constant: $round(x) = ceil(x - 0.5) = floor(x + 0.5)$.

An additional constraint is that weights are represented on a certain number of bits, **b**. This can be achieved by parameterizing the function in the following way:

$$F_b(x) = \frac{ceil(2^{b-1}x)}{2^{b-1}} \tag{2}$$

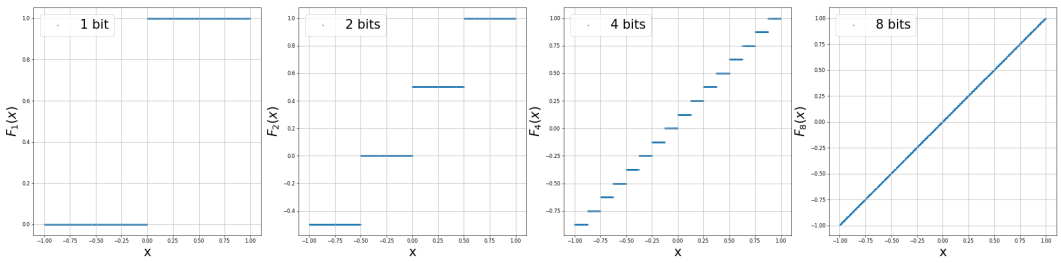

Figure 1: Parameterized ceiling function.

If we consider the interval $[-1, +1]$ we observe that the values of x are mapped to 2^b different values, as shown in Figure 1. For 8 bits $F_b(x)$ looks almost indistinguishable from x. Note that there are

values of x which are mapped by the ceiling function to exactly 0. The lower the number of bits the larger the amount of zero values. We would like to avoid this for the moment because it implies weight pruning which is not of interest in this work. In order to lower the probability of setting weights to exactly zero the quantization function $F_b(x)$ is modified to the following expression:

$$Q_b(x) = \max\left(0, \frac{\lceil (x \cdot 2^{b-1}) \rceil}{2^{b-1}}\right) + \max\left(0, -\frac{\lfloor (x \cdot 2^{b-1}) \rfloor}{2^{b-1}}\right) \cdot \text{sign}(x) \tag{3}$$

where $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ represent the ceil and floor functions, respectively and b represents the number of bits, as before. Figure 2 shows the image of $Q_b(x)$.

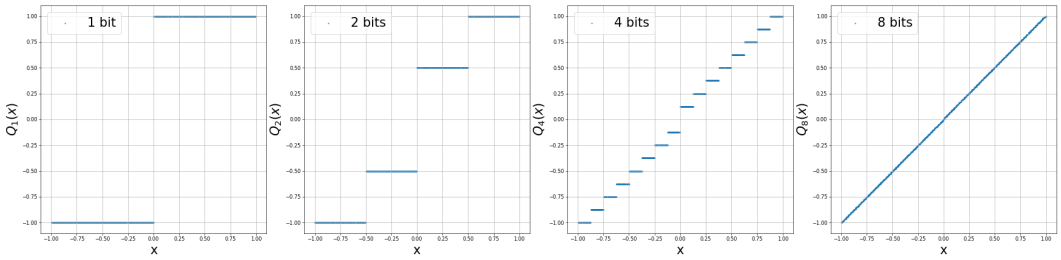

Figure 2: Quantization function.

The ceiling function is used for the positive values of w to "lift" all weights such that $w \geq 1$ and the floor function for the negative values of w to "lower" all weights such that $w \leq -1$. Note that for $b = 1$ the quantization function reduces to the sign function. If weights are represented by 32-bit floating point values, the probability that n weights are exactly zero is 2^{-32n}, therefore we do not consider zero values to be of any concern in this approach.

For good convergence, during training the weights are scaled to the interval $[-\alpha, +\alpha]$, so they become $\hat{w} = \frac{\alpha \cdot w}{\max(|w|)}$. This scalar coefficient is a positive value which, along with the weights of a layer l is a trainable parameter in the network. Following the above considerations we rewrite Eq. (1) by passing the (32-bit floating point) weights through the quantization function $Q_b(x)$:

$$h_j^l = \sigma\left(\sum_{i=0}^{n_{l-1}} |\alpha^l| \cdot Q_b(\hat{w}_{ij}^l) \cdot h_i^{l-1}\right) \tag{4}$$

After the network is trained the obtained weights $\mathbf{W}^l = |\alpha^l| \cdot Q_b(\hat{w}_{ij}^l)$ are all mapped to the set $\left\{-\alpha^l, -\alpha^l \cdot \frac{2^b-1}{2^b}, \cdots, -\frac{\alpha^l}{2^b}, +\frac{\alpha^l}{2^b}, \cdots, \alpha^l \cdot \frac{2^b-1}{2^b}, \alpha^l\right\}$. Therefore dividing all weights by the smallest magnitude in the given layer they map to the interval of signed integers $\left\{-2^b, -2^b - 1, \cdots, -1, +1, \cdots, 2^b - 1, 2^b\right\}$.

Although the quantization function is not differentiable, the straight through estimator [1] can be used as a good approximation of the gradients during backpropagation.

2 Experimental results

The current section is concerned with the classification performance of quantized networks. The architectures of the networks are ResNet-18 [2] adapted for the CIFAR-10 [3] dataset and LeNet300 [4] for MNIST [5].

2.1 Weight quantization

The first experiments considered only training quantized weights, while activations are still on 32-bit floating point values. This serves as a proof of concept that the weight quantization method is valid.

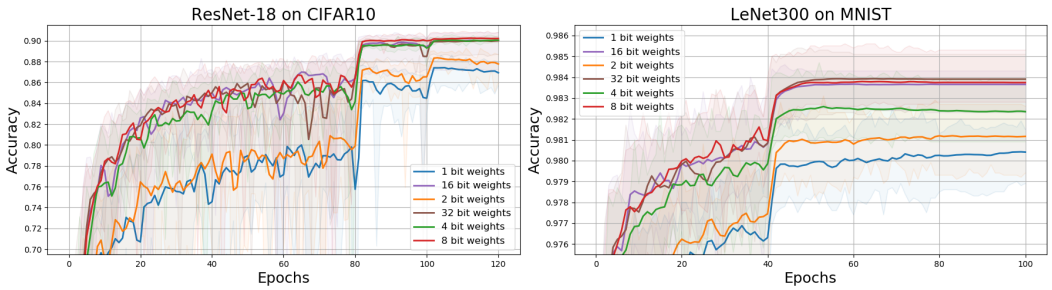

Figure 3: Experiments with weight quantization on ResNet-18 (left) and LeNet300 (right). Each solid curve is the mean of 5 runs. Shaded areas show the minimum and maximum for the 5 runs.

Figure 3 illustrates the classification performance achieved by quantizing the weights of these networks on **1**, **2**, **4**, **8**, **16** and **32** bits (during training). It indicates there is little to no loss of accuracy when switching from 32 to 16 bits and, surprisingly, in these tests for ResNet 8-bit weights seems to perform slightly better than their 32-bit counterparts. The solid curves indicate the mean of 5 independent training runs. Shaded areas indicate the minimum and maximum accuracy for the 5 runs.

Key Takeaways: These experiments show that it is indeed possible to directly train neural networks with quantized weights. It is enough to pass the 32-bit floating point weights through a parameterized quantization function. Moreover, no significant performance degradation is observed even when training with 4-bit weights. Extreme weight quantizations of 2 and 1 bit-depth do show a performance loss, however they are very close to the accuracy of the full precision models: within .5 percentage points for MNIST and 2.5 for CIFAR-10.

2.2 Weight and activation quantization

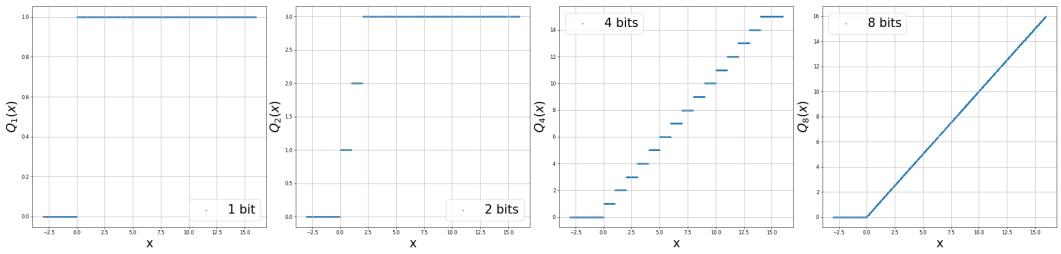

Figure 4: Quantization function from Eq. (5)

The quantization of the activation functions follow a similar approach as for the weights. However, considering ReLU as the activation for neurons, the quantization function has to use **b** bits and map values in the interval $\left[0, 2^b - 1\right]$ simply because the maximum value representable is $2^b - 1$. The sum in Eq. (1) might be larger than $2^b - 1$, therefore it needs to be clipped first in the specified interval: $\hat{x}_b = \mathrm{clip}(x, 0, 2^b - 1)$. Therefore we use the following quantization function:

$$A_b(x) = \mathrm{ReLU}\left(\frac{\max(\hat{x}_b)}{2^b - 1}\left\lfloor\frac{\hat{x}_b(2^b - 1)}{\max(\hat{x}_b)}\right\rceil\right) \tag{5}$$

Figure 4 shows the image of $A_b(x)$ from Eq. (5) for $b \in \{1, 2, 4, 8\}$. Considering the interval $[-3, 16]$, one can observe that the function maps all values of x to 2^b discreet values in the interval $\left[0, 2^b - 1\right]$. Again, as for the weight quantization function, $A_b(x)$ is almost indistinguishable from $\mathrm{ReLU}(x)$ when the number of bits is 8. There are other possible schemes for handling the activation quantization but this work is limited only to Eq. (5).

In order to find the impact of the different quantization schemes on the final performance of the networks, a scan was performed where the networks were trained for several runs checking all quantization combinations. The results are shown in Figure 5.

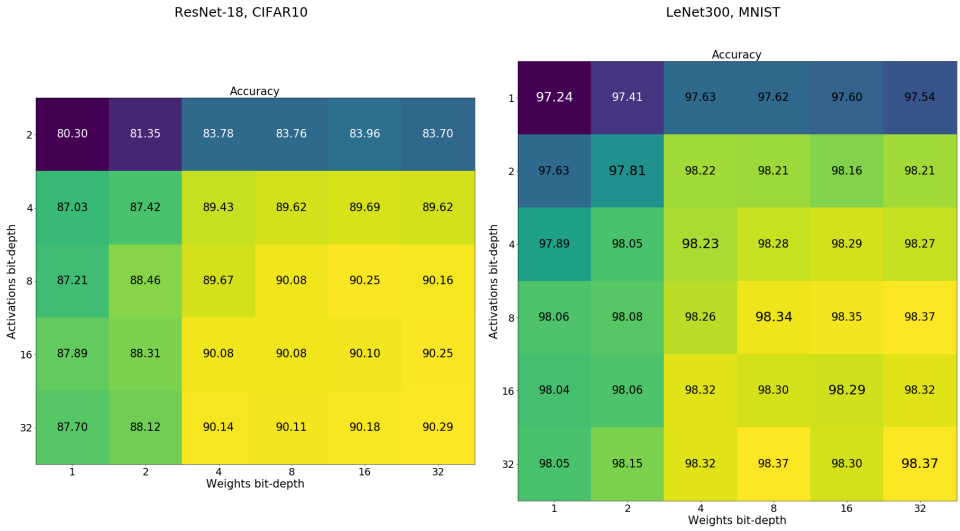

Figure 5: Experiments with weight quantization and activation quantization with LeNet300 on MNIST.

The right panel shows the accuracy of LeNet300 as a function of the bit-depth used for the weights (X-axis) and bit-depth for the activations (Y-axis). The left panels shows the accuracy for ResNet-18 on CIFAR-10. Both plots indicate negligible performance loss when switching from 32 to 16, 8 or 4 bit-depth along both dimensions. The accuracy starts to degrade rapidly when the activations are represented using very low bit-depths. However, additional improvements can be done for both activation schemes which might be able to recover the accuracy drop in these regimes.

Key Takeaway: It is possible to directly train networks where the weights as well as the activations are quantized. Little loss is incurred even when training with 4 bits.

3 Summary

This work proposes a quantization scheme applicable to both weights and activations of a neural network. This scheme scales from a simple fully connected network applied on a toy-dataset like MNIST to more complex residual networks on closer to real-world data like CIFAR-10. It is also very general in the sense that the bit-depth can be arbitrarily specified. It shows how a simple function can be applied to the real-valued weights of a network during training in order to simulate discreet values of arbitrary bit-depth. The experiments indicate that quantizing as low as 4 bits is possible with no significant accuracy loss.

References

[1] Y. Bengio. Estimating or propagating gradients through stochastic neurons. *arXiv*, 1305.2982, 2013.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.