Overcoming Challenges in Fixed Point Training of Deep Convolutional Networks

Darryl D. Lin   DARRYL.DLIN@GMAIL.COM
Qualcomm Research, San Diego, CA 92121 USA

Sachin S. Talathi   TALATHI@GMAIL.COM
Qualcomm Research, San Diego, CA 92121 USA

Abstract

It is known that training deep neural networks, in particular, deep convolutional networks, with aggressively reduced numerical precision is challenging. The stochastic gradient descent algorithm becomes unstable in the presence of noisy gradient updates resulting from arithmetic with limited numeric precision. One of the well-accepted solutions facilitating the training of low precision fixed point networks is stochastic rounding. However, to the best of our knowledge, the source of the instability in training neural networks with noisy gradient updates has not been well investigated. This work is an attempt to draw a theoretical connection between low numerical precision and training algorithm stability. In doing so, we will also propose and verify through experiments methods that are able to improve the training performance of deep convolutional networks in fixed point.

1. Introduction

Deep convolutional networks (DCNs) have demonstrated state-of-the-art performance in many machine learning tasks such as image classification (Krizhevsky et al., 2012) and speech recognition (Deng et al., 2013). However, the complexity and the size of DCNs have limited their use in mobile applications and embedded systems. One reason is related to the hit on performance (in terms of accuracy on a given machine learning task) that these networks take when they are deployed with data representations using reduced numeric precision. A potential avenue to alleviate this problem is to fine-tune pre-trained floating point DCNs using data representations with reduced numeric precision.

However, the training algorithms have a strong tendency to diverge when the precision of network parameters and features are too low (Han et al., 2015; Courbariaux et al., 2014).

More recently, several works have touched upon the issue of training deep networks with low numerical precision (Gupta et al., 2015; Lin et al., 2015; Gysel et al., 2016). In all of these works *stochastic rounding* has been the key to improving the convergence properties of the training algorithm, which in turn has enabled training of deep networks with relatively small bit-widths.

However, to the best of our knowledge, there is a limited understanding from a theoretical point of view as to why low precision networks lead to training difficulties. In this paper, we attempt offer a theoretical insight into the root cause of the numerical instability when training DCNs with limited numeric precision representations. In doing so, we will also propose a few solutions to combat such instability in order to improve the training outcome. These proposals are not meant to replace stochastic rounding. Rather, they are complementary techniques. To clearly demonstrate the effectiveness of our proposed solutions, we will not perform stochastic rounding in the experiments. We intend to combine stochastic rounding and our proposed solutions in future works.

This work will focus on fine-tuning a pre-trained floating point DCN in fixed point. While most of the analysis apply also to the case of training a fixed point network from scratch, some discussions may be applicable to the fixed point fine-tuning scenario alone.

2. Low Precision and Back-Propagation

In this section, we will investigate the origin of instability in the network training phase when low precision weights and activations are used. The outcome of this effort will shed light on possible avenues to alleviate the problem.

2.1. Effective Activation Function

The computation of activations in the forward pass of a deep network can be written as:

$$a_i^{(l)} = \sum_j w_{i,j}^{(l)} \cdot g(a_j^{(l-1)}), \qquad (1)$$

where $a_i^{(l)}$ denotes the i-th activation in the l-th layer, $w_{i,j}^{(l)}$ represents the (i,j)-th weight value in the l-th layer. And $g(\cdot)$ is the activation function.

Note that here we assume both the activations and weights are full precision values. Now consider the case where only the weights are low precision fixed point values. From the forward pass perspective, (1) still holds.

However, when we introduce low precision activations into the equation, (1) is no longer an accurate description of how the activations propagate. To see this, we may consider the evaluation of $a_i^{(l)}$ in fixed point representation as in Figure 1.
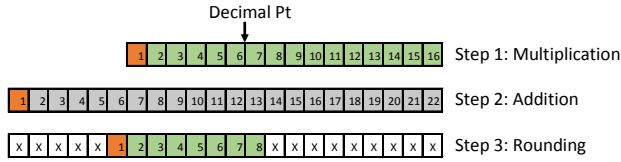
Figure 1. Evaluation of activation as quantization

In Figure 1, three operations are depicted:

- Step 1: Compute $w \cdot g(a)$. Assuming both w and $g(a)$ are 8-bit fixed point values, the product is a 16-bit value.

- Step 2: Compute $\sum w \cdot g(a)$. The size of the accumulator is larger than 16-bit to prevent overflow.

- Step 3: The outcome of $\sum w \cdot g(a)$ is rounded and truncated to produce an 8-bit activation value.

Step 3 is a quantization step that reduces the precision of the value calculated based on (1) in keeping with the desired fixed point precision of layer l. In essence, assuming ReLU, the effective activation function experienced by the features in the network is as shown in Figure 2(b), rather than 2(a).

2.2. Gradient Mismatch

In back-propagation, denoting the cost function as C, an important equation that dictates how the "error signal", $\frac{\partial C}{\partial a_i^{(l)}}$, propagates down the network is expressed as follows:

$$\frac{\partial C}{\partial a_i^{(l)}} = g'(a_i^{(l)}) \cdot \sum_j w_{i,j}^{(l+1)} \frac{\partial C}{\partial a_j^{(l+1)}}. \qquad (2)$$
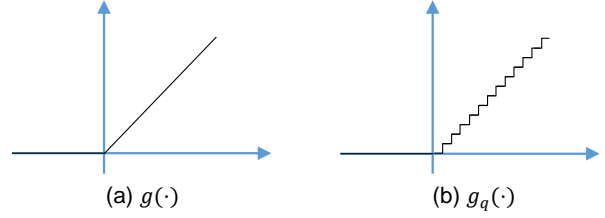
(a) $g(\cdot)$ (b) $g_q(\cdot)$

Figure 2. The presumed and actual ReLU function in low precision networks

The value of $-\frac{\partial C}{\partial a_i^{(l)}}$ indicates the direction in which $a_i^{(l)}$ should move in order to improve the cost function. Playing a crucial role in (2) is the derivative of the activation function, $g'(a_i^{(l)})$. In a software environment that implements SGD, original activation functions in the form of Figure 2(a) is assumed. However, as explained in Section 2.1, the effective activation function in a fixed point network is a non-differentiable function as described in Figure 2(b).

This disagreement between the presumed and the actual activation function is the origin of what we call the "gradient mismatch" problem. When the bit-widths of the weights and activations are large, the gradient of the original activation function offers a good approximation to that of the quantized activation function. However, the mismatch will start to impact the stability of SGD when the bit-widths become too small (step sizes become too large).

The gradient mismatch problem also exacerbates as the error signal propagates deeper down the network, because every time the presumed $g'(a_i^{(l)})$ is used, additional errors are introduced in the gradient computation. Since the gradients w.r.t. the weights are directly based on the gradients w.r.t. the activations,

$$\frac{\partial C}{\partial w_{i,j}^{(l)}} = g(a_i^{(l-1)}) \cdot \frac{\partial C}{\partial a_j^{(l)}}, \qquad (3)$$

the weight updates become increasingly inaccurate as the error propagates into lower layers of the network. Hence training networks in fixed point is much more challenging in deeper networks than in shallower networks.

2.3. Potential Solutions

Having understood the source of the issue, we will propose a few methods to help overcome the challenges of training or fine-tuning a fixed point network. The obvious approach of replacing the perceived activation function with the effective activation function that takes quantization into account is not viable because the effective activation function is not differentiable. However, some alternatives may help improve convergence during model training to avoid the gradient mismatch problem.

2.3.1. PROPOSAL 1: LOW PRECISION WEIGHTS AND FULL PRECISION ACTIVATIONS

Recognizing that the main obstacle of training in fixed point is the low precision activations, we may train a network with the desired precision for the weights, while keeping the activations floating point or with relatively high precision. After training, the network can be adapted to run with lower precision activations.

2.3.2. PROPOSAL 2: FINE-TUNING TOP LAYER(S) ONLY

As the analysis in Section 2 shows, when the activation precision is low, weight updates of top layers are more reliable than lower layers, because the gradient mismatch builds up from the top of the network to the bottom. Therefore, while it may not be possible to fine-tune the entire network, it may be possible to fine-tune only the top layers without incurring convergence issues.

2.3.3. PROPOSAL 3: BOTTOM-TO-TOP ITERATIVE FINE-TUNING

The bottom-to-top iterative fine-tuning scheme is a training algorithm designed to avoid gradient mismatch. At the same time, it allows the entire network to be fine-tuned. For example, consider a network with 4 layers. Table 1 offers an illustration of how fine-tuning is divided into phases where one layer is fine-tuned in each phase.

Table 1. Example showing the phases of iterative fine-tuning

	Phase 1		Phase 2		Phase 3	
	Acts	Wgts	Acts	Wgts	Acts	Wgts
Layer4	Float	-	Float	-	Float	update
Layer3	Float	-	Float	update	FixPt	-
Layer2	Float	update	FixPt	-	FixPt	-
Layer1	FixPt	-	FixPt	-	FixPt	-

Each phase of fine-tuning, consisting of 1 or multiple epochs, updates the weights of one of the layers (weights can follow the desired fixed point format without special treatment). As shown in Table 1, Phase 1 fine-tunes the weights of Layer2. After Phase 1 is complete, Phase 2 fine-tunes the weights of Layer3 while keeping the weights of all other layers static. Then Phase 3 fine-tunes Layer4 in a similar manner. Note that Layer1 weights are quantized but never fine-tuned.

Also of importance is how the number format of activations change over the phases. Initially during Phase 1, only the bottom layer (Layer1) activations are in fixed point, but in Phase 2, both Layer1 and Layer2 activations are in fixed

point. In the last phase of fine-tuning, only the output of the final layer remains floating point. All other activations have been turned into fixed point. The gradual turning on of fixed point activations is designed to prevent gradient mismatch completely. Careful inspection of the algorithm shows that, whenever the weights of a particular layer are updated, the gradients are always back-propagated from layers with only floating point activations.

3. Experiments

In this section, we examine the effectiveness of the proposed solutions based on a deep convolutional network we developed for the ImageNet classification task [1]. The network has 12 convolutional layers and 5 fully-connected layers. We choose this network to experiment because, as we have shown in a network designed for CIFAR-10 classification (Lin et al., 2016), fine-tuning a relatively shallow fixed point network does not pose convergence challenges even when the bit-widths are small.

Table 2. ImageNet classification Top-5 error rate (%): No fine-tuning

Activation	Weight Bit-width			
Bit-width	4	8	16	Float
4	98.6	33.4	32.9	32.7
8	97.1	19.3	18.0	18.2
16	96.6	15.0	14.3	14.4
Float	96.6	14.1	13.9	13.8

The baseline for the experiment is the DCN network that is quantized based on the algorithm presented in Lin et al. (2016) without fine-tuning. The Top-5 error rates of these networks, for different weight and activation bit-width combinations, are listed in Table 2. Note that for all the fixed point experiments in this paper, the output activations of the final fully-connected layer is always set to a bit-width of 16. We do not try to reduce the precision of this quantity because the subsequent softmax layer is rather sensitive to low precision inputs and it is an insignificant overhead to the network overall.

To further improve the accuracy beyond Table 2, we perform fine-tuning on these networks subject to the corresponding fixed point bit-width constraints of the weights and activations. Table 3 shows that, while fine-tuning improves some scenarios (for example, 16-bit activations and 4-bit weights), it fails to converge for most of the settings where the activations are in fixed point. This interesting observation validates the analysis in Section 2 showing that the stability problem is due to the low precision of activa-

---

[1]Proprietary Information, Qualcomm Inc

Table 3. ImageNet classification Top-5 error rate (%): Plain vanilla fine-tuning ("n/a"="fails to converge")

Activation	Weight Bit-width			
Bit-width	4	8	16	Float
4	n/a	n/a	n/a	n/a
8	n/a	19.3	n/a	n/a
16	21.0	n/a	n/a	n/a
Float	22.2	13.5	13.3	13.8

tions, not weights. We note that for these and all the subsequent fine-tuning experiments, we did not perform any hyperparameter optimization of the training parameters and it is quite possible to identify a set of training hyperparameters for which the quantized network may train successfully.

3.1. Proposal 1

Table 4. ImageNet classification Top-5 error rate (%): Use fixed point activations in networks trained with floating point activations (Proposal 1)

Activation	Weight Bit-width			
Bit-width	4	8	16	Float
4	45.6	32.0	31.3	32.7
8	25.1	16.8	16.8	18.2
16	22.5	13.9	13.8	14.4
Float	22.2	13.5	13.3	13.8

The networks on the last row of Table 3 are already trained with the desired weight precision. We can directly use them to run with different activation precision. Table 4 lists the classification accuracy of this approach. It is seen that we can achieve fairly good classification accuracy for different activation bit-widths.

3.2. Proposal 2

Using the networks on the last row of Table 3 as the baseline, we can continue to fine-tune only the weights of the top few layers. It is possible to fine-tune the top layers because the effect of gradient mismatch accumulates toward the lower layers of the network, but the impact on the top layers is relatively small.

Table 5 demonstrates the results of fine-tuning only the top fully-connected layer in the network. It is seen that fine-tuning the top layer offers a small boost in accuracy compared to the networks in Table 4.

Table 5. ImageNet classification Top-5 error rate (%): Fine-tune the top fully-connected layer (Proposal 2)

Activation	Weight Bit-width			
Bit-width	4	8	16	Float
4	37.1	23.8	23.3	23.5
8	22.8	15.6	15.7	16.2
16	21.2	13.7	13.5	13.7
Float	22.2	13.5	13.3	13.8

3.3. Proposal 3

Again using the network on the last row of Table 3 as the fine-tuning baseline, we iteratively fine-tune the network from the bottom to the top, one layer at a time, according to the algorithm prescribed in Table 1. This procedure ensures that each layer has accurate gradient information when the weights are updated.

Table 6. ImageNet classification Top-5 error rate (%): Iterative fine-tuning from bottom layer to top layer (Proposal 3)

Activation	Weight Bit-width			
Bit-width	4	8	16	Float
4	25.3	18.4	18.3	18.2
8	19.3	15.2	14.1	14.1
16	18.8	13.2	13.2	13.5
Float	22.2	13.5	13.3	13.8

As seen in Table 6, this approach provides a significant performance boost compared to the previous solutions. Even a network with 4-bit weights and 4-bit activations is able to achieve Top-5 error rate of 25.3%. Some of the entries in the table have better accuracy than the floating point baseline. This may be attributed to the regularization effect of the added quantization noisy during training (Lin et al., 2015).

4. Conclusion

In this paper, we studied the effect of low numerical precision of weights and activations on the accuracy of gradient computation during back-propagation. Our analysis showed that low precision weights are benign, but low precision activations have a detrimental impact on the computed gradients. The errors in gradient computation accumulate during back-propagation and may slow and even prevent the successful convergence of gradient descent when the network is sufficiently deep.

We proposed a few solutions to combat this problem and demonstrated through experiments their effectiveness on the ImageNet classification task. We plan to combine

stochastic rounding and our proposed solutions in future works.

References

Courbariaux, M., Bengio, Y., and David, J. Low precision arithmetic for deep learning. *arXiv:1412.7024*, 2014.

Deng, L., G.E., Hinton, and Kingsbury, B. New types of deep neural network learning for speech recognition and related applications: an overview. In *IEEE International Conference on Acoustic, Speech and Signal Processing*, pp. 8599–8603, 2013.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. *arXiv:1502.02551*, 2015.

Gysel, P., Motamedi, M., and Ghiasi, S. Hardware-oriented approximation of convolutional neural networks. *arXiv:1604.03168*, 2016.

Han, S., Mao, H., and Dally, W. J. A deep neural network compression pipeline: Pruning, quantization, Huffman encoding. *arXiv:1510.00149*, 2015.

Krizhevsky, A., Sutskever, I., and Hinton, G.E. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

Lin, D. D., Talathi, S. S., and Annapureddy, V. S. Fixed point quantization of deep convolutional networks. In *ICML*, 2016.

Lin, Z., Courbariaux, M., Memisevic, R., and Bengio, Y. Neural networks with few multiplications. *arXiv:1510.03009*, 2015.