Analytical Guarantees on Numerical Precision of Deep Neural Networks

Charbel Sakr Yongjune Kim Naresh Shanbhag

Abstract

The acclaimed successes of neural networks often overshadow their tremendous complexity. We focus on numerical precision - a key parameter defining the complexity of neural networks. First, we present theoretical bounds on the accuracy in presence of limited precision. Interestingly, these bounds can be computed via the back-propagation algorithm. Hence, by combining our theoretical analysis and the back-propagation algorithm, we are able to readily determine the minimum precision needed to preserve accuracy without having to resort to time-consuming fixed-point simulations. We provide numerical evidence showing how our approach allows us to maintain high accuracy but with lower complexity than state-of-the-art binary networks.

1. Introduction

Neural networks have achieved state-of-the-art accuracy on many machine learning tasks. AlexNet (Krizhevsky et al., 2012) had a deep impact a few years ago in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and triggered intensive research efforts on deep neural networks. Recently, ResNet (He et al., 2016) has outperformed humans in recognition tasks.

These networks have very high computational complexity. For instance, AlexNet has 60 million parameters and 650,000 neurons (Krizhevsky et al., 2012). Its convolutional layers alone require 666 million multiply-accumulates (MACs) per 227×227 image ($13k$ MACs/pixel) and 2.3 million weights (Chen et al., 2016). Deepface's network involves more than 120 million parameters (Taigman et al., 2014). ResNet is a 152-layer deep residual network. This high complexity of deep neural networks prevents its deployment on energy and resource-constrained platforms such as mobile devices and autonomous platforms.

1.1. Related Work

One of the most effective approaches for reducing resource utilization is to implement fixed-point neural networks. As mentioned in (Lin et al., 2016a), there are two approaches for designing fixed-point neural networks: (1) directly train a fixed-point neural network, and (2) quantize a pre-trained floating-point neural network to obtain a fixed-point network.

As an example of fixed-point training, Gupta et al. (2015) showed that 16-bit fixed-point representation incurs little accuracy degradation by using stochastic rounding. A more aggressive approach is to design binary networks such as Kim & Smaragdis (2016) which used bitwise operations to replace the arithmetic operations and Rastegari et al. (2016) which explored optimal binarization schemes. BinaryConnect (Courbariaux et al., 2015) trained networks using binary weights while BinaryNet (Hubara et al., 2016b) trained networks with binary weights and activations.

Although these fixed-point training approaches make it possible to design fixed-point neural networks achieving excellent accuracy, training based on fixed-point arithmetic is generally harder than floating-point training since the optimization is done in a discrete space.

Hence, in this paper, we focus on the second approach that quantizes a pre-trained floating-pointing network to a fixed-point network. This approach leverages the extensive work in training state-of-the-art floating-point neural networks such as dropout (Srivastava et al., 2014), maxout (Goodfellow et al., 2013), network-in-network (Lin et al., 2013), and residual learning (He et al., 2016) to name a few. In this approach, proper precision needs to be determined after training to reduce complexity while minimizing the accuracy loss. In (Hwang & Sung, 2014), exhaustive search is performed to determine a suitable precision allocation. Recently, Lin et al. (2016a) offered an analytical solution for non-uniform bit precision based on the signal-to-quantization-noise ratio (SQNR). However, the use of non-uniform quantization step sizes at each layer is diffi-

The authors are with the University of Illinois at Urbana-Champaign, 1308 W Main St., Urabna, IL 61801 USA. Correspondence to: Charbel Sakr <sakr2@illinois.edu>, Yongjune Kim <yongjune@illinois.edu>, Naresh Shanbhag <shanbhag@illinois.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

cult to implement as it requires multiple variable precision arithmetic units.

In addition to fixed-point implementation, many approaches have been proposed to lower the complexity of deep neural networks in terms of the number of arithmetic operations. Han et al. (2015) employs a three-step training method to identify important connections, prune the unimportant ones, and retrain on the pruned network. Zhang et al. (2015) replaces the original convolutional layers by smaller sequential layers to reduce computations. These approaches are complementary to our technique of quantizing a pre-trained floating-point neural network into a fixed-point one.

In this paper, we obtain analytical bounds on the accuracy of fixed-point networks that are obtained by quantizing a conventionally trained floating-point network. Furthermore, by defining meaningful measures of a fixed-point network's hardware complexity viz. computational and representational costs, we develop a principled approach to precision assignment using these bounds in order to minimize these complexity measures.

1.2. Contributions

Our contributions are both theoretical and practical. We summarize our main contributions:

- We derive *theoretical bounds* on the misclassification rate in presence of limited precision and thus determine *analytically how accuracy and precision trade-off with each other*.
- Employing the theoretical bounds and the back-propagation algorithm, we show that *proper precision assignments* can be readily determined while *maintaining accuracy close to floating-point networks*.
- We analytically determine which of weights or activations need more precision, and we show that typically *the precision requirements of weights are greater than those of activations* for fully-connected networks and are similar to each other for networks with shared weights such as convolutional neural networks.
- We introduce *computational and representational costs* as meaningful metrics to evaluate the complexity of neural networks under fixed precision assignment.
- We validate our findings on the MNIST and CIFAR10 datasets demonstrating the ease with which fixed-point networks with *complexity smaller than state-of-the-art binary networks* can be derived from pre-trained floating-point networks with minimal loss in accuracy.

It is worth mentioning that our proposed method is general and can be applied to every class of neural networks such as multilayer perceptrons and convolutional neural networks.

2. Fixed-Point Neural Networks

2.1. Accuracy of Fixed-Point and Floating-Point Networks

For a given floating-point neural network and its fixed-point counterpart we define: 1) the floating-point error probability $p_{e,fl} = \Pr\{\hat{Y}_{fl} \neq Y\}$ where $\hat{Y}_{fl}$ is the output of the floating-point network and Y is the true label; 2) the fixed-point error probability $p_{e,fx} = \Pr\{\hat{Y}_{fx} \neq Y\}$ where $\hat{Y}_{fx}$ is the output of the fixed-point network; 3) the mismatch probability between fixed-point and floating-point $p_m = \Pr\{\hat{Y}_{fx} \neq \hat{Y}_{fl}\}$. Observe that:

$$p_{e,fx} \leq p_{e,fl} + p_m \tag{1}$$

The right-hand-side represents the worst case of having no overlap between misclassified samples and samples whose predicted labels are in error due to quantization. We provide a formal proof of (1) in the supplementary section. Note that $p_{e,fx}$ is a quantity of interest as it characterizes the accuracy of the fixed-point system. We employ p_m as a proxy to $p_{e,fx}$ because it brings in the effects of quantization into the picture as opposed to $p_{e,fl}$ which solely depends on the algorithm. This observation was made in (Sakr et al., 2017) and allowed for an analytical characterization of linear classifiers as a function of precision.

2.2. Fixed-Point Quantization

The study of fixed-point systems and algorithms is well established in the context of signal processing and communication systems (Shanbhag, 2016). A popular example is the least mean square (LMS) algorithm for which bounds on precision requirements for input, weights, and updates have been derived (Goel & Shanbhag, 1998). In such analyses, it is standard practice (Caraiscos & Liu, 1984) to assume all signed quantities lie in $[-1, 1]$ and all unsigned quantities lie in $[0, 2]$. Of course, activations and weights can be designed to satisfy this assumption during training. A B-bit fixed-point number a_{fx} would be related to its floating-point value a as follows:

$$a_{fx} = a + q_a \tag{2}$$

where q_a is the quantization noise which is modeled as an independent uniform random variable distributed over $\left[-\frac{\Delta}{2}, \frac{\Delta}{2}\right]$, where $\Delta = 2^{-(B-1)}$ is the quantization step (Caraiscos & Liu, 1984).

2.3. Complexity in Fixed-Point

We argue that the complexity of a fixed-point system has two aspects: computational and representational costs. In what follows, we consider activations and weights to be quantized to B_A and B_W bits, respectively.

The *computational cost* is a measure of the computational resources utilized for generating a single decision, and is defined as the number of 1 bit full adders (FAs). A full adder is a canonical building block of arithmetic units. We assume arithmetic operations are executed using the commonly used ripple carry adder (Knauer, 1989) and Baugh-Wooley multiplier (Baugh & Wooley, 1973) architectures designed using FAs. Consequently, the number of FAs used to compute a D-dimensional dot product of activations and weights is (Lin et al., 2016b):

$$DB_AB_W + (D-1)(B_A + B_W + \lceil \log_2(D) \rceil - 1) \quad (3)$$

Hence, an important aspect of the computational cost of a dot product is that it is an increasing function of the *product* of activation precision (B_A), weight precision (B_W), and dimension (D).

We define the *representational cost* as the total number of bits needed to represent all network parameters, i.e., both activations and weights. This cost is a measure of the storage complexity and communications costs associated with data movement. The total representational cost of a fixed-point network is:

$$|\mathcal{A}|\, B_A + |\mathcal{W}|\, B_W \quad (4)$$

bits, where $\mathcal{A}$ and $\mathcal{W}$ are the index sets of all activations and weights in the network, respectively. Observe that the representational cost is *linear* in activation and weight precisions as compared to the computational cost.

Equations (3) - (4) illustrate that, though computational and representational costs are not independent, they are different. Together, they describe the implementation costs associated with a network. We shall employ both when evaluating the complexity of fixed-point networks.

2.4. Setup

Here, we establish notation. Let us consider neural networks deployed on a M-class classification task. For a given input, the network would typically have M numerical outputs $\{z_i\}_{i=1}^M$ and the decision would be $\hat{y} = \arg\max_{i=1,\ldots,M} z_i$. Each numerical output is a function of weights and activations in the network:

$$z_i = f\left(\{a_h\}_{h\in\mathcal{A}}, \{w_h\}_{h\in\mathcal{W}}\right) \quad (5)$$

for $i = 1, \ldots, M$, where a_h denotes the activation indexed by h and w_h denotes the weight indexed by h. When activations and weights are quantized to B_A and B_W bits, respectively, the output z_i is corrupted by quantization noise q_{z_i} so that:

$$z_i + q_{z_i} = f\left(\{a_h + q_{a_h}\}_{h\in\mathcal{A}}, \{w_h + q_{w_h}\}_{h\in\mathcal{W}}\right) \quad (6)$$

where q_{a_h} and q_{w_h} are the quantization noise terms of the activation a_h and weight w_h, respectively. Here, $\{q_{a_h}\}_{h\in\mathcal{A}}$ are independent uniformly distributed random variables on $\left[-\frac{\Delta_A}{2}, \frac{\Delta_A}{2}\right]$ and $\{q_{w_h}\}_{h\in\mathcal{W}}$ are independent uniformly distributed random variables on $\left[-\frac{\Delta_W}{2}, \frac{\Delta_W}{2}\right]$, with $\Delta_A = 2^{-(B_A-1)}$ and $\Delta_W = 2^{-(B_W-1)}$.

In quantization noise analysis, it is standard to ignore cross-products of quantization noise terms as their contribution is negligible. Therefore, using Taylor's theorem, we express the total quantization noise at the output of the fixed-point network as:

$$q_{z_i} = \sum_{h\in\mathcal{A}} q_{a_h} \frac{\partial z_i}{\partial a_h} + \sum_{h\in\mathcal{W}} q_{w_h} \frac{\partial z_i}{\partial w_h}. \quad (7)$$

Note that the derivatives in (7) are obtained as part of the back-propagation algorithm. Thus, using our results, it is possible to estimate the precision requirements of deep neural networks during training itself. As will be shown later, this requires one additional back-propagation iteration to be executed after the weights have converged.

3. Bounds on Mismatch Probability

3.1. Second Order Bound

We present our first result. It is an analytical upper bound on the mismatch probability p_m between a fixed-point neural network and its floating-point counterpart.

Theorem 1. *Given B_A and B_W, the mismatch probability p_m between a fixed-point network and its floating-point counterpart is upper bounded as follows:*

$$p_m \leq \frac{\Delta_A^2}{24}\mathbb{E}\left[\sum_{\substack{i=1\\ i\neq \hat{Y}_{fl}}}^{M} \frac{\sum_{h\in\mathcal{A}} \left|\frac{\partial(Z_i - Z_{\hat{Y}_{fl}})}{\partial A_h}\right|^2}{|Z_i - Z_{\hat{Y}_{fl}}|^2}\right] + \frac{\Delta_W^2}{24}\mathbb{E}\left[\sum_{\substack{i=1\\ i\neq \hat{Y}_{fl}}}^{M} \frac{\sum_{h\in\mathcal{W}} \left|\frac{\partial(Z_i - Z_{\hat{Y}_{fl}})}{\partial w_h}\right|^2}{|Z_i - Z_{\hat{Y}_{fl}}|^2}\right] \quad (8)$$

where expectations are taken over a random input and $\{A_h\}_{h\in\mathcal{A}}$, $\{Z_i\}_{i=1}^M$, and $\hat{Y}_{fl}$ are thus random variables.

Proof. The detailed proof can be found in the supplementary section. Here, we provide the main idea and the intuition behind the proof.

The heart of the proof lies in evaluating $\Pr\left(z_i + q_{z_i} > z_j + q_{z_j}\right)$ for any pair of outputs z_i and z_j where $z_j > z_i$. Equivalently, we need to evaluate

$\Pr\left(q_{z_i} - q_{z_j} > z_j - z_i\right)$. But from (7), we have:

$$q_{z_i} - q_{z_j} = \sum_{h\in\mathcal{A}} q_{a_h} \frac{\partial(z_i - z_j)}{\partial a_h} + \sum_{h\in\mathcal{W}} q_{w_h} \frac{\partial(z_i - z_j)}{\partial w_h}. \tag{9}$$

In (9), we have a linear combination of quantization noise terms, $q_{z_i} - q_{z_j}$ is a zero mean random variable having a symmetric distribution. This means that $\Pr\left(q_{z_i} - q_{z_j} > z_j - z_i\right) = \frac{1}{2}\Pr\left(|q_{z_i} - q_{z_j}| > |z_j - z_i|\right)$, which allows us to use Chebyshev's inequality. Indeed, from (9), the variance of $q_{z_i} - q_{z_j}$ is given by:

$$\frac{\Delta_A^2}{12}\sum_{h\in\mathcal{A}}\left|\frac{\partial(z_i - z_j)}{\partial a_h}\right|^2 + \frac{\Delta_W^2}{12}\sum_{h\in\mathcal{W}}\left|\frac{\partial(z_i - z_j)}{\partial w_h}\right|^2,$$

so that

$$\Pr\left(z_i + q_{z_i} > z_j + q_{z_j}\right) \leq \frac{\Delta_A^2 \sum_{h\in\mathcal{A}}\left|\frac{\partial(z_i - z_j)}{\partial a_h}\right|^2 + \Delta_W^2 \sum_{h\in\mathcal{W}}\left|\frac{\partial(z_i - z_j)}{\partial w_h}\right|^2}{24\left|z_i - z_j\right|^2}. \tag{10}$$

As explained in the supplementary section, it is possible to obtain to (8) from (10) using standard probabilistic arguments. □

Before proceeding, we point out that the two expectations in (8) are taken over a random input but the weights $\{w_h\}_{h\in\mathcal{W}}$ are frozen after training and are hence deterministic.

Several observations are to be made. First notice that the mismatch probability p_m increases with Δ_A^2 and Δ_W^2. This is to be expected as smaller precision leads to more mismatch. Theorem 1 says a little bit more: the mismatch probability decreases exponentially with precision, because $\Delta_A = 2^{-(B_A-1)}$ and $\Delta_W = 2^{-(B_W-1)}$.

Note that the quantities in the expectations in (8) can be obtained as part of a standard back-propagation procedure. Indeed, once the weights are frozen, it is enough to perform one forward pass on an estimation set (which should have statistically significant cardinality), record the numerical outputs, perform one backward pass and probe all relevant derivatives. Thus, (8) can be readily computed.

Another practical aspect of Theorem 1 is that this operation needs to be done only once as these quantities do not depend on precision. Once they are determined, for any given precision assignment, we simply evaluate (8) and combine it with (1) to obtain an estimate (upper bound) on the accuracy of the fixed-point instance. This way the precision necessary to achieve a specific mismatch probability is obtained from a trained floating-point network. This clearly highlights the gains in practicality of our analytical approach over a trial-and-error based search.

Finally, (8) reveals a very interesting aspect of the trade-off between activation precision B_A and weight precision B_W. We rewrite (8) as:

$$p_m \leq \Delta_A^2 E_A + \Delta_W^2 E_W \tag{11}$$

where

$$E_A = \mathbb{E}\left[\sum_{\substack{i=1\\ i\neq \hat{Y}_{fl}}}^{M} \frac{\sum_{h\in\mathcal{A}}\left|\frac{\partial(Z_i - Z_{\hat{Y}_{fl}})}{\partial A_h}\right|^2}{24|Z_i - Z_{\hat{Y}_{fl}}|^2}\right]$$

and

$$E_W = \mathbb{E}\left[\sum_{\substack{i=1\\ i\neq \hat{Y}_{fl}}}^{M} \frac{\sum_{h\in\mathcal{W}}\left|\frac{\partial(Z_i - Z_{\hat{Y}_{fl}})}{\partial w_h}\right|^2}{24|Z_i - Z_{\hat{Y}_{fl}}|^2}\right].$$

The first term in (11) characterizes the impact of quantizing activations on the overall accuracy while the second characterizes that of weight quantization. It might be the case that one of the two terms dominates the sum depending on the values of E_A and E_W. This means that either the activations or the weights are assigned more precision than necessary. An intuitive first step to efficiently get a smaller upper bound is to make the two terms of comparable order. That can be made by setting $\Delta_A^2 E_A = \Delta_W^2 E_W$ which is equivalent to

$$B_A - B_W = round\left(\log_2\sqrt{\frac{E_A}{E_W}}\right) \tag{12}$$

where $round()$ denotes the rounding operation. This is an effective way of taking care of one of the two degrees of freedom introduced by (8).

A natural question to ask would be which of E_A and E_W is typically larger. That is to say, to whom, activations or weights, should one assign more precision. In deep neural networks, there are more weights than activations, a trend particularly observed in deep networks with most layers fully connected. This trend, though not as pronounced, is also observed in networks with shared weights, such as convolutional neural networks. However, there exist a few counterexamples such as the networks in (Hubara et al., 2016b) and (Hubara et al., 2016a). It is thus reasonable to expect $E_W \geq E_A$, and consequently *the precision requirements of weights will, in general, be more than those of activations.*

One way to interpret (11) is to consider minimizing the upper bound in (8) subject to $B_A + B_W = c$ for some constant

c. Indeed, it can be shown that (12) would be a necessary condition of the corresponding solution. This is an application of the arithmetic-geometric mean inequality. Effectively, (11) is of particular interest when considering computational cost which increases as a function of the product of both precisions (see Section 2.3).

3.2. Tighter Bound

We present a tighter upper bound on p_m based on the Chernoff bound.

Theorem 2. *Given B_A and B_W, the mismatch probability p_m between a fixed-point network and its floating-point counterpart is upper bounded as follows:*

$$p_m \leq \mathbb{E}\left[\sum_{\substack{i=1\\ i\neq \hat{Y}_{fl}}}^{M} e^{-S^{(i,\hat{Y}_{fl})}} P_1^{(i,\hat{Y}_{fl})} P_2^{(i,\hat{Y}_{fl})}\right] \tag{13}$$

where, for $i \neq j$,

$$S^{(i,j)} = \frac{3(Z_i - Z_j)^2}{\sum_{h\in\mathcal{A}}\left(D_{A_h}^{(i,j)}\right)^2 + \sum_{h\in\mathcal{W}}\left(D_{w_h}^{(i,j)}\right)^2},$$

$$D_{A_h}^{(i,j)} = \frac{\Delta_A}{2}\frac{\partial(Z_i - Z_j)}{\partial A_h}, \quad D_{w_h}^{(i,j)} = \frac{\Delta_W}{2}\frac{\partial(Z_i - Z_j)}{\partial w_h},$$

$$P_1^{(i,j)} = \prod_{h\in\mathcal{A}} \frac{\sinh\left(T^{(i,j)} D_{A_h}^{(i,j)}\right)}{T^{(i,j)} D_{A_h}^{(i,j)}},$$

$$P_2^{(i,j)} = \prod_{h\in\mathcal{W}} \frac{\sinh\left(T^{(i,j)} D_{w_h}^{(i,j)}\right)}{T^{(i,j)} D_{w_h}^{(i,j)}},$$

and

$$T^{(i,j)} = \frac{S^{(i,j)}}{Z_j - Z_i}.$$

Proof. Again, we leave the technical details for the supplementary section. Here we also provide the main idea and intuition.

As in Theorem 1, we shall focus on evaluating $\Pr\left(z_i + q_{z_i} > z_j + q_{z_j}\right) = \Pr\left(q_{z_i} - q_{z_j} > z_j - z_i\right)$ for any pair of outputs z_i and z_j where $z_j > z_i$. The key difference here is that we will use the Chernoff bound in order to account for the complete quantization noise statistics. Indeed, letting $v = z_j - z_i$, we have:

$$\Pr\left(q_{z_i} - q_{z_j} > v\right) \leq e^{-tv}\mathbb{E}\left[e^{t(q_{z_i} - q_{z_j})}\right]$$

for any $t > 0$. We show that:

$$\mathbb{E}\left[e^{t(q_{z_i} - q_{z_j})}\right] = \prod_{h\in\mathcal{A}} \frac{\sinh(td_{a,h})}{td_{a,h}} \prod_{h\in\mathcal{W}} \frac{\sinh(td_{w,h})}{td_{w,h}}$$

where $d_{a,h} = \frac{\Delta_A}{2}\frac{\partial(z_i - z_j)}{\partial a_h}$ and $d_{w,h} = \frac{\Delta_W}{2}\frac{\partial(z_i - z_j)}{\partial w_h}$. This yields:

$$\Pr\left(q_{z_i} - q_{z_j} > v\right) \leq e^{-tv} \prod_{h\in\mathcal{A}} \frac{\sinh(td_{a,h})}{td_{a,h}} \prod_{h\in\mathcal{W}} \frac{\sinh(td_{w,h})}{td_{w,h}}. \tag{14}$$

We show that the right-hand-side is minimized over positive values of t when:

$$t = \frac{3v}{\sum_{h\in\mathcal{A}}(d_{a,h})^2 + \sum_{h\in\mathcal{W}}(d_{w,h})^2}.$$

Substituting this value of t into (14) and using standard probabilistic arguments, we obtain (13). □

The first observation to be made is that Theorem 2 indicates that, on average, p_m is upper bounded by an exponentially decaying function of the quantity $S^{(i,\hat{Y}_{fl})}$ for all $i \neq \hat{Y}_{fl}$ up to a correction factor $P_1^{(i,\hat{Y}_{fl})} P_2^{(i,\hat{Y}_{fl})}$. This correction factor is a product of terms typically centered around 1 (each term is of the form $\frac{\sinh(x)}{x} \approx 1$ for small x). On the other hand, $S^{(i,\hat{Y}_{fl})}$, by definition, is the ratio of the excess confidence the floating-point network has in the label $\hat{Y}_{fl}$ over the total quantization noise variance reflected at the output, i.e., $S^{(i,\hat{Y}_{fl})}$ is the SQNR. Hence, Theorem 2 states that the tolerance of a neural network to quantization is, on average, exponentially decaying with the SQNR at its output. In terms of precision, Theorem 2 states that p_m is bounded by a double exponentially decaying function of precision (that is an exponential function of an exponential function). Note how this bound is tighter than that of Theorem 1.

This double exponential relationship between accuracy and precision is not too surprising when one considers the problem of binary hypothesis testing under additive Gaussian noise (Blahut, 2010) scenario. In this scenario, it is well-known that the probability of error is an exponentially decaying function of the signal-to-noise ratio (SNR) in the high-SNR regime. Theorem 2 points out a similar relationship between accuracy and precision but it does so using rudimentary probability principles without relying on high-SNR approximations.

While Theorem 2 is much tighter than Theorem 1 theoretically, it is not as convenient to use. In order to use Theorem 2, one has to perform a forward-backward pass and select relevant quantities and apply (13) for each choice of B_A and B_W. However, a lot of information, e.g. the derivatives, can be reused at each run, and so the runs may be lumped into one forward-backward pass. In a sense, the complexity of computing the bound in Theorem 2 lies between the evaluation of (11) and the complicated conventional trial-and-error based search.

We now illustrate the applications of these bounds.

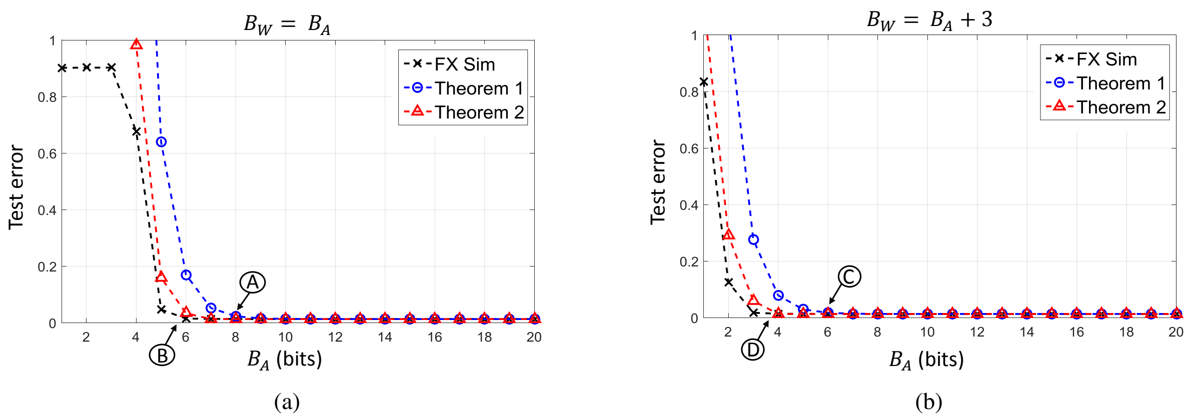

Figure 1: Validity of bounds for MNIST when: (a) $B_W = B_A$ and (b) $B_W = B_A + 3$ as dictated by (12) ($E_A = 41$ and $E_W = 3803$ so that $\log_2 \sqrt{\frac{E_W}{E_A}} \approx 3.2$). FX Sim denotes fixed point simulations.

4. Simulation Results

We conduct numerical simulations to illustrate both the validity and usefulness of the analysis developed in the previous section. We show how it is possible to reduce precision in an aggressive yet principled manner. We present results on two popular datasets: MNIST and CIFAR-10. The metrics we address are threefold:

- Accuracy measured in terms of test error.
- Computational cost measured in $\#FAs$ (see Section 2.3, (3) was used to compute $\#FAs$ per MAC).
- Representational cost measured in bits (see Section 2.3, (4) was used).

We compare our results to similar works conducting similar experiments: 1) the work on fixed-point training with stochastic quantization (SQ) (Gupta et al., 2015) and 2) BinaryNet (BN) (Hubara et al., 2016b).

4.1. DNN on MNIST

First, we conduct simulations on the MNIST dataset for handwritten character recognition (LeCun et al., 1998). The dataset consists of 60K training samples and 10K test samples. Each sample consists of an image and a label. Images are of size 28×28 pixels representing a handwritten digit between 0 and 9. Labels take the value of the corresponding digit.

In this first experiment, we chose an architecture of $784 - 512 - 512 - 512 - 10$, i.e., 3 hidden layers, each of 512 units. We first trained the network in floating-point using the back-propagation algorithm. We used a batch size of 200 and a learning rate of 0.1 with a decay rate of 0.978 per epoch. We restore the learning rate every 100 epochs, the decay rate makes the learning rate vary between 0.1 and 0.01. We train the first 300 epochs using 15% dropout, the second 300 epochs using 20% dropout, and the third 300 epochs using 25% dropout (900 epochs overall). It appears from the original dropout work (Srivastava et al., 2014) that the typical 50% dropout fraction works best for very wide multi-layer perceptrons (MLPs) (4096 to 8912 hidden units). For this reason, we chose to experiment with smaller dropout fractions.

The only pre-processing done is to scale the inputs between -1 and 1. We used ReLU activations with the subtle addition of a right rectifier for values larger than 2 (as discussed in Section 2). The resulting activation is also called a hard sigmoid. We also clipped the weights to lie in $[-1, 1]$ at each iteration. The resulting test error we obtained in floating-point is 1.36%.

Figure 1 illustrates the validity of our analysis. Indeed, both bounds (based on Theorems 1 & 2) successfully upper bound the test error obtained through fixed-point simulations. Figure 1 (b) demonstrates the utility of (12). Indeed, setting $B_W = B_A$ allows us to reduce the precision to about 6 or 7 bits before the accuracy start degrading. In addition, under these conditions we found $E_A = 41$ and $E_W = 3803$ so that $\log_2 \sqrt{\frac{E_W}{E_A}} \approx 3.2$. Thus, setting $B_W = B_A + 3$ as dictated by (12) allows for more aggressive precision reduction. Activation precision B_A can now be reduced to about 3 or 4 bits before the accuracy degrades. To compute the bounds, we used an estimation set of 1000 random samples from the dataset.

We compare our results with SQ which used a $784-1000-1000-10$ architecture on 16-bit fixed-point activations and weights. A stochastic rounding scheme was used to compensate for quantization. We also compare our results with BN with a $784 - 2048 - 2048 - 2048 - 10$ architecture on binary quantities. A stochastic rounding scheme was also used during training.

Table 1 shows some comparisons with related works in terms of accuracy, computational cost, and representational

Precision Assignment	Test error (%)	Computational Cost (10^6 FAs)	Representational Cost (10^6 bits)
Floating-point	1.36	N/A	N/A
$(8, 8)$	1.41	82.9	7.5
$(6, 6)$	1.54	53.1	**5.63**
$(6, 9)$	**1.35**	72.7	8.43
$(4, 7)$	1.43	**44.7**	6.54
SQ $(16, 16)$ (Gupta et al., 2015)	1.4	533	28
BN $(1, 1)$ (Hubara et al., 2016b)	1.4	117	10

Table 1: Results for MNIST: Comparison of accuracy, computational cost, and representational cost with state-of-the-art related works. Chosen precision assignments are obtained from Figure 1.

cost. For comparison, we selected four notable design options from Figures 1 (a,b):

A. Smallest (B_A, B_W) such that $B_W = B_A$ and $p_m \leq 1\%$ as bounded by Theorem 1. In this case $(B_A, B_W) = (8, 8)$.
B. Smallest (B_A, B_W) such that $B_W = B_A$ and $p_m \leq 1\%$ as bounded by Theorem 2. In this case $(B_A, B_W) = (6, 6)$.
C. Smallest (B_A, B_W) such that $B_W = B_A + 3$ as dictated by (12) and $p_m \leq 1\%$ as bounded by Theorem 1. In this case $(B_A, B_W) = (6, 9)$.
D. Smallest (B_A, B_W) such that $B_W = B_A + 3$ as dictated by (12) and $p_m \leq 1\%$ as bounded by Theorem 2. In this case $(B_A, B_W) = (4, 7)$.

As can be seen in Table 1, the accuracy is similar across all design options including the results reported by SQ and BN. Interestingly, for all four design options, our network has a *smaller computational cost than BN*. In addition, SQ's computational cost is about $4.6\times$ that of BN (533M/117M). The greatest reduction in computational cost is obtained for a precision assignment of $(4, 7)$ corresponding to a $2.6\times$ and $11.9\times$ reduction compared to BN (117M/44.7M) and SQ (533M/44.7M), respectively. The corresponding test error rate is of 1.43%. Similar trends are observed for representational costs. Again, our four designs have *smaller representational cost than even BN*. BN itself has $2.8\times$ smaller representational cost than SQ (28M/10M). Note that a precision assignment of $(6, 6)$ yields $1.8\times$ and $5.0\times$ smaller representational costs than BN (10M/5.63M) and SQ (28M/5.63M), respectively. The corresponding test error rate is 1.54%.

The fact that we are able to achieve lesser computational and representational costs than BN while maintaining similar accuracy highlights two important points. First, the width of a network severely impacts its complexity. We made our network four times as narrow as BN's and still managed to use eight times as many bits per parameter without exceeding BN's complexity. Second, our results illustrate the strength of numbering systems, specifically, the strength of fixed-point representations. Our results indicate that a correct and meaningful multi-bit representation of parameters is better in both complexity and accuracy than a 1-bit unstructured allocation.

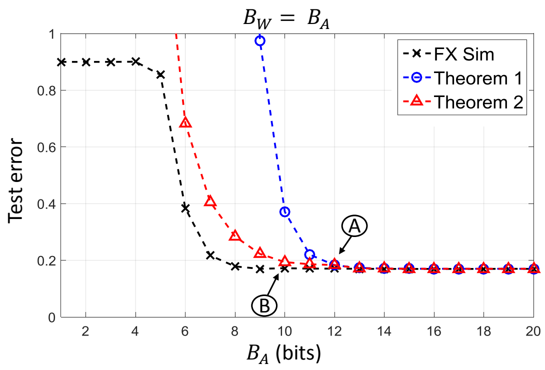

Figure 2: Validity of bounds for CIFAR when $B_W = B_A$ which is also dictated by (12) ($E_A = 21033$ and $E_W = 31641$ so that $\log_2 \sqrt{\frac{E_W}{E_A}} \approx 0.29$). FX Sim denotes fixed point simulations.

4.2. CNN on CIFAR 10

We conduct a similar experiment on the CIFAR10 dataset (Krizhevsky & Hinton, 2009). The dataset consists of 60K color images each representing airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships, and trucks. 50K of these images constitute the training set, and the 10K remaining are for testing. SQ's architecture on this dataset is a simple one: three convolutional layers, interleaved by max pooling layers. The output of the final pooling layer is fed to a 10-way softmax output layer. The reported accuracy using 16-bit fixed-point arithmetic is a 25.4% test error. BN's architecture is a much wider and deeper architecture based on VGG (Simonyan & Zisserman, 2014). The reported accuracy of the binary network is an impressive 10.15% which is of benchmarking quality even for full precision networks.

We adopt a similar architecture as SQ, but leverage re-

Precision Assignment	Test error (%)	Computational Cost (10^6 FAs)	Representational Cost (10^6 bits)
Floating-point	17.02	N/A	N/A
$(12, 12)$	17.08	3626	5.09
$(10, 10)$	17.23	**2749**	**4.24**
SQ $(16, 16)$ (Gupta et al., 2015)	25.4	4203	4.54
BN $(1, 1)$ (Hubara et al., 2016b)	**10.15**	3608	6.48

Table 2: Results for CIFAR10: Comparison of accuracy, computational cost, and representational cost with state-of-the-art related works. Chosen precision assignments are obtained from Figure 2.

cent advances in convolutional neural networks (CNNs) research. It has been shown that adding networks within convolutional layers (in the 'Network in Network' sense) as described in (Lin et al., 2013) significantly enhances accuracy, while not incurring much complexity overhead. Hence, we replace SQ's architecture by a deep one which we describe as $64C5 - 64C1 - 64C1 - MP2 - 64C5 - 64C1 - 64C1 - MP2 - 64C5 - 64FC - 64FC - 64FC - 10$, where $C5$ denotes 5×5 kernels, $C1$ denotes 1×1 kernels (they emulate the networks in networks), $MP2$ denotes 2×2 max pooling, and FC denotes fully connected components. As is customary for this dataset, we apply zero-phase component analysis (ZCA) whitening to the data before training. Because this dataset is a challenging one, we first fine-tune the hyperparameters (learning rate, weight decay rate, and momentum), then train for 300 epochs. The best accuracy we reach in floating point using this 12-layer deep network is 17.02%.

Figure 2 shows the results of our fixed-point simulation and analysis. Note that, while both bounds from Theorems 1 and 2 still successfully upper bound the test error, these are not as tight as in our MNIST experiment. Furthermore, in this case, (12) dictates keeping $B_W = B_A$ as $E_A = 21033$ and $E_W = 31641$ so that $\log_2 \sqrt{\frac{E_W}{E_A}} \approx 0.29$. The fact that $E_W \geq E_A$ is expected as there are typically more weights than activations in a neural network. However, note that in this case the contrast between E_W and E_A is not as sharp as in our MNIST experiment. This is mainly due to the higher weight to activation ratio in fully connected DNNs than in CNNs.

We again select two design options:

A. Smallest (B_A, B_W) such that $B_W = B_A$ and $p_m \leq 1\%$ as bounded by Theorem 1. In this case $(B_A, B_W) = (12, 12)$.

B. Smallest (B_A, B_W) such that $B_W = B_A$ and $p_m \leq 1\%$ as bounded by Theorem 2. In this case $(B_A, B_W) = (10, 10)$.

Table 2 indicates that BN is the most accurate with 10.15% test error. Interestingly, it has lesser computational cost but more representational cost than SQ. This is due to the dependence of the computational cost on the *product* of B_A and B_W. The least complex network is ours when setting $(B_A, B_W) = (10, 10)$ and its test error is 17.23% which is already a large improvement on SQ in spite of having smaller computational and representational costs. This network is also less complex than that of BN.

The main take away here is that CNNs are quite different from fully connected DNNs when it comes to precision requirements. Furthermore, from Table 2 we observe that BN achieves the least test error. It seems that this better accuracy is due to its greater representational power rather than its computational power (BN's representational cost is much higher than the others as opposed to its computational cost).

5. Conclusion

In this paper we analyzed the quantization tolerance of neural networks. We used our analysis to efficiently reduce weight and activation precisions while maintaining similar fidelity as the floating-point initiation. Specifically, we obtained bounds on the mismatch probability between a fixed-point network and its floating-point counterpart in terms of precision. We showed that a neural network's accuracy degradation due to quantization decreases double exponentially as a function of precision. Our analysis provides a straightforward method to obtain an upper bound on the network's error probability as a function of precision. We used these results on real datasets to minimize the computational and representational costs of a fixed-point network while maintaining accuracy.

Our work addresses the general problem of resource constrained machine learning. One take away is that it is imperative to understand the trade-offs between accuracy and complexity. In our work, we used precision as a parameter to analytically characterize this trade-off. Nevertheless, additional aspects of complexity in neural networks such as their structure and their sparsity can also be accounted for. In fact, more work can be done in that regard. Our work may be viewed as a first step in developing a unified and principled framework to understand complexity vs. accuracy trade-offs in deep neural networks and other machine learning algorithms.

Acknowledgment

This work was supported in part by Systems on Nanoscale Information fabriCs (SONIC), one of the six SRC STARnet Centers, sponsored by MARCO and DARPA.

References

Baugh, Charles R and Wooley, Bruce A. A two's complement parallel array multiplication algorithm. *IEEE Transactions on Computers*, 100(12):1045–1047, 1973.

Blahut, Richard E. *Fast algorithms for signal processing*. Cambridge University Press, 2010.

Caraiscos, Christos and Liu, Bede. A roundoff error analysis of the lms adaptive algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(1):34–41, 1984.

Chen, Y. et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 262–263. IEEE, 2016.

Courbariaux, Matthieu et al. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pp. 3123–3131, 2015.

Goel, M. and Shanbhag, N. Finite-precision analysis of the pipelined strength-reduced adaptive filter. *Signal Processing, IEEE Transactions on*, 46(6):1763–1769, 1998.

Goodfellow, Ian J et al. Maxout networks. *ICML (3)*, 28: 1319–1327, 2013.

Gupta, S. et al. Deep Learning with Limited Numerical Precision. In *Proceedings of The 32nd International Conference on Machine Learning*, pp. 1737–1746, 2015.

Han, Song et al. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1135–1143, 2015.

He, Kaiming et al. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

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

Hubara, Itay et al. Binarized neural networks. In *Advances in Neural Information Processing Systems*, pp. 4107–4115, 2016b.

Hwang, Kyuyeon and Sung, Wonyong. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pp. 1–6. IEEE, 2014.

Kim, M. and Smaragdis, P. Bitwise Neural Networks. *arXiv preprint arXiv:1601.06071*, 2016.

Knauer, Karl. Ripple-carry adder, June 13 1989. US Patent 4,839,849.

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

Krizhevsky, Alex et al. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

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

Lin, Darryl et al. Fixed point quantization of deep convolutional networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 2849–2858, 2016a.

Lin, Min et al. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Lin, Yingyan et al. Variation-tolerant architectures for convolutional neural networks in the near threshold voltage regime. In *Signal Processing Systems (SiPS), 2016 IEEE International Workshop on*, pp. 17–22. IEEE, 2016b.

Rastegari, Mohammad et al. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pp. 525–542. Springer, 2016.

Sakr, Charbel et al. Minimum precision requirements for the SVM-SGD learning algorithm. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 2017.

Shanbhag, Naresh R. Energy-efficient machine learning in silicon: A communications-inspired approach. *arXiv preprint arXiv:1611.03109*, 2016.

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

Srivastava, Nitish et al. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Taigman, Yaniv et al. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1701–1708, 2014.

Zhang, Xiangyu et al. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2015.