Convolutional Neural Networks using Logarithmic Data Representation

Daisuke Miyashita DAISUKEM@STANFORD.EDU
Stanford University, Stanford, CA 94305 USA
Toshiba, Kawasaki, Japan

Edward H. Lee EDHLEE@STANFORD.EDU
Stanford University, Stanford, CA 94305 USA

Boris Murmann MURMANN@STANFORD.EDU
Stanford University, Stanford, CA 94305 USA

Abstract

Recent advances in convolutional neural networks have considered model complexity and hardware efficiency to enable deployment onto embedded systems and mobile devices. For example, it is now well-known that the arithmetic operations of deep networks can be encoded down to 8-bit fixed-point without significant deterioration in performance. However, further reduction in precision down to as low as 3-bit fixed-point results in significant losses in performance. In this paper we propose a new data representation that enables state-of-the-art networks to be encoded to 3 bits with negligible loss in classification performance. To perform this, we take advantage of the fact that the weights and activations in a trained network naturally have non-uniform distributions. Using non-uniform, base-2 logarithmic representation to encode weights, communicate activations, and perform dot-products enables networks to 1) achieve higher classification accuracies than fixed-point at the same resolution and 2) eliminate bulky digital multipliers. Finally, we propose an end-to-end training procedure that uses $\log$ representation at 5-bits, which achieves higher final test accuracy than linear at 5-bits.

1. Introduction

Deep convolutional neural networks (CNN) have demonstrated state-of-the-art performance in image classification (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; He et al., 2015) but have steadily grown in computational complexity. For example, the Deep Residual Learning (He et al., 2015) set a new record in image classification accuracy at the expense of 11.3 billion floating-point multiply-and-add operations per forward-pass of an image and 230 MB of memory to store the weights in its 152-layer network.

In order for these large networks to run in real-time applications such as for mobile or embedded platforms, it is often necessary to use low-precision arithmetic and apply compression techniques. Recently, many researchers have successfully deployed networks that compute using 8-bit fixed-point representation (Vanhoucke et al., 2011; Abadi et al., 2015) and have successfully trained networks with 16-bit fixed point (Gupta et al., 2015). This work in particular is built upon the idea that algorithm-level noise tolerance of the network can motivate simplifications in hardware complexity.

Interesting directions point towards matrix factorization (Denton et al., 2014) and tensorification (Novikov et al., 2015) by leveraging structure of the fully-connected (FC) layers. Another promising area is to prune the FC layer before mapping this to sparse matrix-matrix routines in GPUs (Han et al., 2015b). However, many of these inventions aim at systems that meet some required and specific criteria such as networks that have many, large FC layers or accelerators that handle efficient sparse matrix-matrix arithmetic. And with network architectures currently pushing towards increasing the depth of convolutional layers by settling for fewer dense FC layers (He et al., 2015; Szegedy et al., 2015), there are potential problems in motivating a one-size-fits-all solution to handle these computational and memory demands.

We propose a general method of representing and computing the dot products in a network that can allow networks with minimal constraint on the layer properties to run more efficiently in digital hardware. In this paper we explore the use of communicating activations, storing weights, and computing the atomic dot-products in the binary logarithmic (base-2 logarithmic) domain for both inference and training. The motivations for moving to this domain are the following:

- Training networks with weight decay leads to final weights that are distributed non-uniformly around 0.
- Similarly, activations are also highly concentrated near 0. Our work uses rectified Linear Units (ReLU) as the non-linearity.
- Logarithmic representations can encode data with very large dynamic range in fewer bits than can fixed-point representation (Gautschi et al., 2016).
- Data representation in $\log$-domain is naturally encoded in digital hardware (as shown in Section 4.3).

Our contributions are listed:

- we show that networks obtain higher classification accuracies with logarithmic quantization than linear quantization using traditional fixed-point at equivalent resolutions.
- we show that activations are more robust to quantization than weights. This is because the number of activations tend to be larger than the number of weights which are reused during convolutions.
- we apply our logarithmic data representation on state-of-the-art networks, allowing activations and weights to use only 3b with almost no loss in classification performance.
- we generalize base-2 arithmetic to handle different base. In particular, we show that a base-$\sqrt{2}$ enables the ability to capture large dynamic ranges of weights and activations but also finer precisions across the encoded range of values as well.
- we develop logarithmic backpropagation for efficient training.

2. Related work

Reduced-precision computation. (Shin et al., 2016; Sung et al., 2015; Vanhoucke et al., 2011; Han et al., 2015a) analyzed the effects of quantizing the trained weights for inference. For example, (Han et al., 2015b) shows that convolutional layers in AlexNet (Krizhevsky et al., 2012) can be encoded to as little as 5 bits without a significant accuracy penalty. There has also been recent work in training using low precision arithmetic. (Gupta et al., 2015) propose a stochastic rounding scheme to help train networks using 16-bit fixed-point. (Lin et al., 2015) propose quantized back-propagation and ternary connect. This method reduces the number of floating-point multiplications by casting these operations into powers-of-two multiplies, which are easily realized with bitshifts in digital hardware. They apply this technique on MNIST and CIFAR10 with little loss in performance. However, their method does not completely eliminate all multiplications end-to-end. During test-time the network uses the learned full resolution weights for forward propagation. Training with reduced precision is motivated by the idea that high-precision gradient updates is unnecessary for the stochastic optimization of networks (Bottou & Bousquet, 2007; Bishop, 1995; Audhkhasi et al., 2013). In fact, there are some studies that show that gradient noise helps convergence. For example, (Neelakantan et al., 2015) empirically finds that gradient noise can also encourage faster exploration and annealing of optimization space, which can help network generalization performance.

Hardware implementations. There have been a few but significant advances in the development of specialized hardware of large networks. For example (Farabet et al., 2010) developed Field-Programmable Gate Arrays (FPGA) to perform real-time forward propagation. These groups have also performed a comprehensive study of classification performance and energy efficiency as function of resolution. (Zhang et al., 2015) have also explored the design of convolutions in the context of memory versus compute management under the RoofLine model. Other works focus on specialized, optimized kernels for general purpose GPUs (Chetlur et al., 2014).

3. Concept and Motivation

Each convolutional and fully-connected layer of a network performs matrix operations that distills down to dot products $y = w^T x$, where $x \in \mathbb{R}^n$ is the input, $w \in \mathbb{R}^n$ the weights, and y the activations before being transformed by the non-linearity (e.g. ReLU). Using conventional digital hardware, this operation is performed using n multiply-and-add operations using floating or fixed point representation as shown in Figure 1(a). However, this dot product can also be computed in the $\log$-domain as shown in Figure 1(b,c).

3.1. Proposed Method 1.

The first proposed method as shown in Figure 1(b) is to transform one operand to its $\log$ representation, convert the resulting transformation back to the linear domain, and

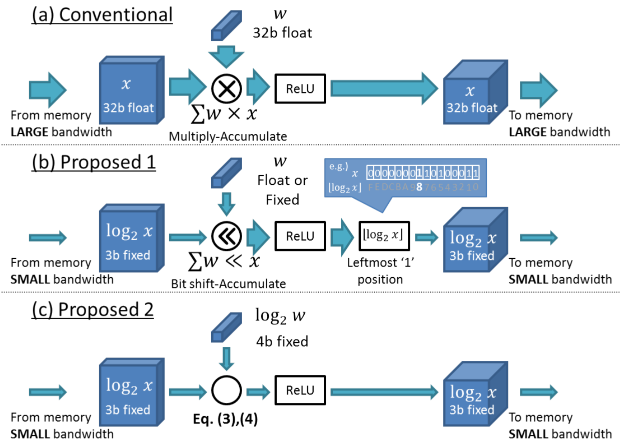

Figure 1. Concept and motivation of this study.

multiply this by the other operand. This is simply

$$
\begin{aligned}
w^T x &\simeq \sum_{i=1}^{n} w_i \times 2^{\tilde{x}_i} \\
&= \sum_{i=1}^{n} \text{Bitshift}(w_i, \tilde{x}_i), \qquad (1)
\end{aligned}
$$

where $\tilde{x}_i = \text{Quantize}(\log_2(x_i))$, $\text{Quantize}(\bullet)$ quantizes $\bullet$ to an integer, and $\text{Bitshift}(a, b)$ is the function that bitshifts a value a by an integer b in fixed-point arithmetic. In floating-point, this operation is simply an addition of b with the exponent part of a. Taking advantage of the $\text{Bitshift}(a, b)$ operator to perform multiplication obviates the need for expensive digital multipliers.

Quantizing the activations and weights in the log-domain ($\log_2(x)$ and $\log_2(w)$) instead of x and w is also motivated by leveraging structure of the non-uniform distributions of x and w. A detailed treatment is shown in the next section. In order to quantize, we propose two hardware-friendly flavors. The first option is to simply floor the input. This method computes $\lfloor \log_2(w) \rfloor$ by returning the position of the first 1 bit seen from the most significant bit (MSB). The second option is to round to the nearest integer, which is more precise than the first option. With the latter option, after computing the integer part, the fractional part is computed in order to assert the rounding direction. This method of rounding is summarized as follows. Pick m bits followed by the leftmost 1 and consider it as a fixed point number F with 0 integer bit and m fractional bits. Then, if $F \geq \sqrt{2} - 1$, round F up to the nearest integer and otherwise round it down to the nearest integer.

3.2. Proposed Method 2.

The second proposed method as shown in Figure 1(c) is to extend the first method to compute dot products in the log-domain for both operands. Additions in linear-domain map to sums of exponentials in the log-domain and multiplications in linear become log-addition. The resulting dot-product is

$$
\begin{aligned}
w^T x &\simeq \sum_{i=1}^{n} 2^{\text{Quantize}(\log_2(w_i)) + \text{Quantize}(\log_2(x_i))} \\
&= \sum_{i=1}^{n} \text{Bitshift}(1, \tilde{w}_i + \tilde{x}_i), \qquad (2)
\end{aligned}
$$

where the log-domain weights are $\tilde{w}_i = \text{Quantize}(\log_2(w_i))$ and log-domain inputs are $\tilde{x}_i = \text{Quantize}(\log_2(x_i))$.

By transforming both the weights and inputs, we compute the original dot product by bitshifting 1 by an integer result $\tilde{w}_i + \tilde{x}_i$ and summing over all i.

3.3. Accumulation in $\log$ domain

Although Fig. 1(b,c) indicates a logarithm-to-linear converter between layers where the actual accumulation is performed in the linear domain, this accumulation is able to be performed in the log-domain using the approximation $\log_2(1 + x) \simeq x$ for $0 \leq x < 1$. For example, let $s_n = w_1 x_1 + \ldots + w_n x_n$, $\tilde{s}_n = \log_2(s_n)$, and $\tilde{p}_i = \tilde{w}_i + \tilde{x}_i$. When $n = 2$,

$$
\begin{aligned}
\tilde{s}_2 &= \log_2 \left(\sum_{i=1}^{2} \text{Bitshift}\left(1, \tilde{p}_i\right) \right) \\
&\simeq \max\left(\tilde{p}_1, \tilde{p}_2\right) + \text{Bitshift}\left(1, -|\tilde{p}_1 - \tilde{p}_2|\right), \qquad (3)
\end{aligned}
$$

and for n in general,

$$
\tilde{s}_n \simeq \max\left(\tilde{s}_{n-1}, \tilde{p}_n\right) + \text{Bitshift}\left(1, -|\lfloor \tilde{s}_{n-1} \rfloor - \tilde{p}_n|\right). \qquad (4)
$$

Note that $\tilde{s}_i$ preserves the fractional part of the word during accumulation. Both accumulation in linear domain and accumulation in $\log$ domain have its pros and cons. Accumulation in linear domain is simpler but requires larger bit widths to accommodate large dynamic range numbers. Accumulation in $\log$ in (3) and (4) appears to be more complicated, but is in fact simply computed using bit-wise operations in digital hardware.

4. Experiments of Proposed Methods

Here we evaluate our methods as detailed in Sections 3.1 and 3.2 on the classification task of ILSVRC-2012 (Deng

Table 1. Structure of AlexNet(Krizhevsky et al., 2012) with quantization

layer	# Weight	# Input	FSR
ReLU(Conv1)	$96 \cdot 3 \cdot 11^2$	$3 \cdot 227^2$	-
LogQuant1	-	$96 \cdot 55^2$	$\mathrm{fsr} + 3$
LRN1	-	-	-
Pool1	-	$96 \cdot 55^2$	-
ReLU(Conv2)	$256 \cdot 96 \cdot 5^2$	$96 \cdot 27^2$	-
LogQuant2	-	$256 \cdot 27^2$	$\mathrm{fsr} + 3$
LRN2	-	-	-
Pool2	-	$256 \cdot 27^2$	-
ReLU(Conv3)	$384 \cdot 256 \cdot 3^2$	$256 \cdot 13^2$	-
LogQuant3	-	$384 \cdot 13^2$	$\mathrm{fsr} + 4$
ReLU(Conv4)	$384 \cdot 384 \cdot 3^2$	$384 \cdot 13^2$	-
LogQuant4	-	$384 \cdot 13^2$	$\mathrm{fsr} + 3$
ReLU(Conv5)	$256 \cdot 384 \cdot 3^2$	$384 \cdot 13^2$	-
LogQuant5	-	$256 \cdot 13^2$	$\mathrm{fsr} + 3$
Pool5	-	$256 \cdot 13^2$	-
ReLU(FC6)	$4096 \cdot 256 \cdot 6^2$	$256 \cdot 6^2$	-
LogQuant6	-	4096	$\mathrm{fsr} + 1$
ReLU(FC7)	$4096 \cdot 4096$	4096	-
LogQuant7	-	4096	fsr
FC8	$1000 \cdot 4096$	4096	-

et al., 2009) using Chainer (Tokui et al., 2015). We evaluate method 1 (Section 3.1) on inference (forward pass) in Section 4.1. Similarly, we evaluate method 2 (Section 3.2) on inference in Sections 4.2 and 4.3. For those experiments, we use published models (AlexNet (Krizhevsky et al., 2012), VGG16 (Simonyan & Zisserman, 2014)) from the caffe model zoo ((Jia et al., 2014)) without any fine tuning (or extra retraining). Finally, we evaluate method 2 on training in Section 4.4.

4.1. Logarithmic Representation of Activations

This experiment evaluates the classification accuracy using logarithmic activations and floating point 32b for the weights. In similar spirit to that of (Gupta et al., 2015), we describe the logarithmic quantization layer **LogQuant** that performs the element-wise operation as follows:

$$\mathrm{LogQuant}(x, \mathrm{bitwidth}, \mathrm{FSR}) = \begin{cases} 0 & x = 0, \\ 2^{\tilde{x}} & \text{otherwise}, \end{cases} \quad (5)$$

where

$$\tilde{x} = \mathrm{Clip}\left(\mathrm{Round}(\log_2(|x|)), \mathrm{FSR} - 2^{\mathrm{bitwidth}}, \mathrm{FSR}\right), \quad (6)$$

$$\mathrm{Clip}(x, \min, \max) = \begin{cases} 0 & x \leq \min, \\ \max - 1 & x \geq \max, \\ x & \text{otherwise}. \end{cases} \quad (7)$$

Table 2. Structure of VGG16(Simonyan & Zisserman, 2014) with quantization

layer	# Weight	# Input	FSR
ReLU(Conv1_1)	$64 \cdot 3 \cdot 3^2$	$3 \cdot 224^2$	-
LogQuant1_1	-	$64 \cdot 224^2$	$\mathrm{fsr} + 4$
ReLU(Conv1_2)	$64 \cdot 64 \cdot 3^2$	$64 \cdot 224^2$	-
LogQuant1_2	-	$64 \cdot 224^2$	$\mathrm{fsr} + 6$
Pool1	-	$64 \cdot 224^2$	-
ReLU(Conv2_1)	$128 \cdot 64 \cdot 3^2$	$64 \cdot 112^2$	-
LogQuant2_1	-	$128 \cdot 112^2$	$\mathrm{fsr} + 6$
ReLU(Conv2_2)	$128 \cdot 128 \cdot 3^2$	$128 \cdot 112^2$	-
LogQuant2_2	-	$128 \cdot 112^2$	$\mathrm{fsr} + 7$
Pool2	-	$128 \cdot 112^2$	-
ReLU(Conv3_1)	$256 \cdot 128 \cdot 3^2$	$128 \cdot 56^2$	-
LogQuant3_1	-	$256 \cdot 56^2$	$\mathrm{fsr} + 7$
ReLU(Conv3_2)	$256 \cdot 256 \cdot 3^2$	$256 \cdot 56^2$	-
LogQuant3_2	-	$256 \cdot 56^2$	$\mathrm{fsr} + 7$
ReLU(Conv3_3)	$256 \cdot 256 \cdot 3^2$	$256 \cdot 56^2$	-
LogQuant3_3	-	$256 \cdot 56^2$	$\mathrm{fsr} + 7$
Pool3	-	$256 \cdot 56^2$	-
ReLU(Conv4_1)	$512 \cdot 256 \cdot 3^2$	$256 \cdot 28^2$	-
LogQuant4_1	-	$512 \cdot 28^2$	$\mathrm{fsr} + 7$
ReLU(Conv4_2)	$512 \cdot 512 \cdot 3^2$	$512 \cdot 28^2$	-
LogQuant4_2	-	$512 \cdot 28^2$	$\mathrm{fsr} + 6$
ReLU(Conv4_3)	$512 \cdot 512 \cdot 3^2$	$512 \cdot 28^2$	-
LogQuant4_3	-	$512 \cdot 28^2$	$\mathrm{fsr} + 5$
Pool4	-	$512 \cdot 28^2$	-
ReLU(Conv5_1)	$512 \cdot 512 \cdot 3^2$	$512 \cdot 14^2$	-
LogQuant5_1	-	$512 \cdot 14^2$	$\mathrm{fsr} + 4$
ReLU(Conv5_2)	$512 \cdot 512 \cdot 3^2$	$512 \cdot 14^2$	-
LogQuant5_2	-	$512 \cdot 14^2$	$\mathrm{fsr} + 3$
ReLU(Conv5_3)	$512 \cdot 512 \cdot 3^2$	$512 \cdot 14^2$	-
LogQuant5_3	-	$512 \cdot 14^2$	$\mathrm{fsr} + 2$
Pool5	-	$512 \cdot 14^2$	-
ReLU(FC6)	$4096 \cdot 512 \cdot 7^2$	$512 \cdot 7^2$	-
LogQuant6	-	4096	$\mathrm{fsr} + 1$
ReLU(FC7)	$4096 \cdot 4096$	4096	-
LogQuant7	-	4096	fsr
FC8	$1000 \cdot 4096$	4096	-

These layers perform the logarithmic quantization and computation as detailed in Section 3.1. Tables 1 and 2 illustrate the addition of these layers to the models. The quantizer has a specified full scale range, and this range in linear scale is 2^{FSR}, where we express this as simply FSR throughout this paper for notational convenience. The FSR values for each layer are shown in Tables 1 and 2; they show fsr added by an offset parameter. This offset parameter is chosen to properly handle the variation of activation ranges from layer to layer using 100 images from the training set. The fsr is a parameter which is global to the network and is tuned to perform the experiments to measure the effect of FSR on classification accuracy. The $\mathrm{bitwidth}$ is the number of bits required to represent a number after quantization. Note that since we assume applying quantization after ReLU function, x is 0 or positive and then we use unsigned format without sign bit for activations.

In order to evaluate our logarithmic representation, we detail an equivalent linear quantization layer described as

$$\begin{aligned}&\mathrm{LinearQuant}(x, \mathrm{bitwidth}, \mathrm{FSR}) \\ &= \mathrm{Clip}\left(\mathrm{Round}\left(\frac{x}{\mathrm{step}}\right) \times \mathrm{step}, 0, 2^{\mathrm{FSR}}\right)\end{aligned} \tag{8}$$

and where

$$\mathrm{step} = 2^{\mathrm{FSR}-\mathrm{bitwidth}}. \tag{9}$$

Figure 2 illustrates the effect of the quantizer on activations following the conv2_2 layer used in VGG16. The pre-quantized distribution tends to 0 exponentially, and the log-quantized distribution illustrates how the log-encoded activations are uniformly equalized across many output bins which is not prevalent in the linear case. Many smaller activation values are more finely represented by log quantization compared to linear quantization. The total quantization error $\frac{1}{N}||\mathrm{Quantize}(x) - x||_1$, where $\mathrm{Quantize}(\bullet)$ is $\mathrm{LogQuant}(\bullet)$ or $\mathrm{LinearQuant}(\bullet)$, x is the vectorized activations of size N, is less for the log-quantized case than for linear. This result is illustrated in Figure 3. Using linear quantization with step size of 1024, we obtain a distribution of quantization errors that are highly concentrated in the region where $|\mathrm{LinearQuant}(x) - x| < 512$. However, log quantization with the bitwidth as linear results in a significantly lower number of quantization errors in the region $128 < |\mathrm{LogQuant}(x) - x| < 512$. This comes at the expense of a slight increase in errors in the region $512 < |\mathrm{LogQuant}(x) - x|$. Nonetheless, the quantization errors $\frac{1}{N}||\mathrm{LogQuant}(x) - x||_1 = 34.19$ for log and $\frac{1}{N}||\mathrm{LogQuant}(x) - x||_1 = 102.89$ for linear.

We run the models as described in Tables 1 and 2 and test on the validation set without data augmentation. We evaluate it with variable bitwidths and FSRs for both quantizer layers.

Figure 4 illustrates the results of AlexNet. Using only 3 bits to represent the activations for both logarithmic and linear quantizations, the top-5 accuracy is still very close to that of the original, unquantized model encoded at floating-point 32b. However, logarithmic representations tolerate a large dynamic range of FSRs. For example, using 4b log, we can obtain 3 order of magnitude variations in the full scale without a significant loss of top-5 accuracy. We see similar results for VGG16 as shown in Figure 5. Table 3 lists the classification accuracies with the optimal FSRs for each case. There are some interesting observations. First, 3b log performs 0.2% worse than 3b linear for AlexNet but 6.2% better for VGG16, which is a higher capacity network than AlexNet. Second, by encoding the activations in 3b log, we achieve the same top-5 accuracy compared to that achieved by 4b linear for VGG16. Third, with 4b log, there is no loss in top-5 accuracy from the original float32 representation.

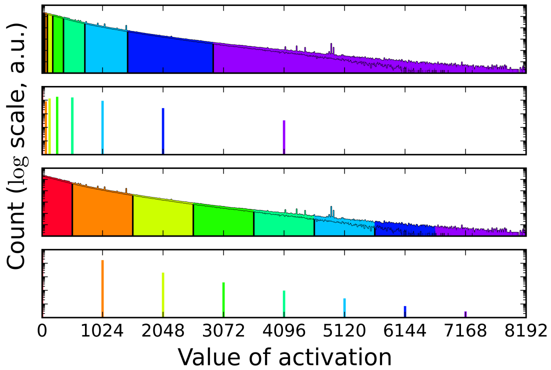

Figure 2. Distribution of activations of conv2_2 layer in VGG16 before and after log and linear quantization. The order (from top to bottom) is: before log-quantization, after log-quantization, before linear quantization, and after linear quantization. The color highlights the binning process of these two quantizers.

Table 3. Top-5 accuracies with quantized activations at optimal FSRs

Model	AlexNet	VGG16
Float 32b	78.3%	89.8%
Log. 3b	76.9%(fsr = 7)	89.2%(fsr = 6)
Log. 4b	76.9%(fsr = 15)	89.8%(fsr = 11)
Linear 3b	77.1%(fsr = 5)	83.0%(fsr = 3)
Linear 4b	77.6%(fsr = 5)	89.4%(fsr = 4)

4.2. Logarithmic Representation of Weights of Fully Connected Layers

The FC weights are quantized using the same strategies as those in Section 4.1, except that they have sign bit. We evaluate the classification performance using log data representation for both FC weights and activations jointly using method 2 in Section 3.2. For comparison, we use linear for FC weights and log for activations as reference. For both methods, we use optimal 4b log for activations that were computed in Section 4.1.

Table 4 compares the mentioned approaches along with floating point. We observe a small 0.4% win for log over linear for AlexNet but a 0.2% decrease for VGG16. Nonetheless, log computation is performed without the use of multipliers.

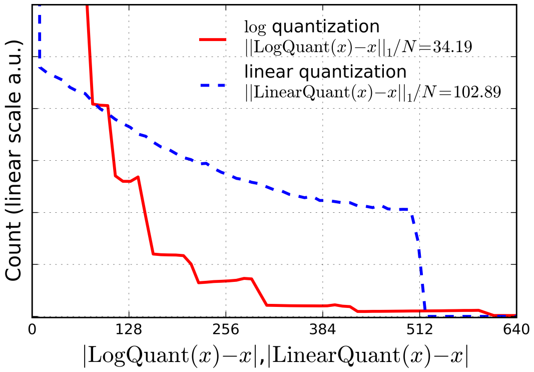

Figure 3. Comparison of the quantization error distribution between logarithmic quantization and linear quantization

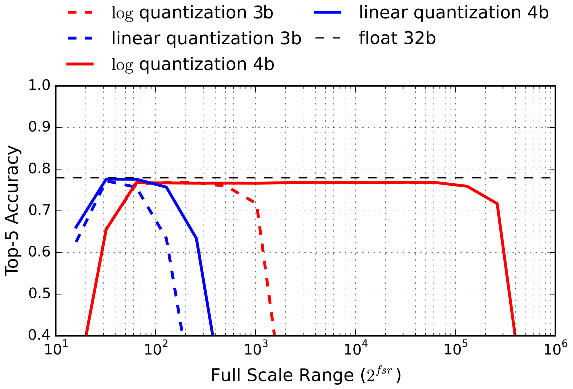

Figure 4. Top5 Accuracy vs Full scale range: AlexNet

Table 4. Top-5 accuracy after applying quantization to weights of FC layers

Model	Float 32b	Log. 4b	Linear 4b
AlexNet	76.9%	76.8%	76.4%
VGG16	89.8%	89.5%	89.7%

An added benefit to quantization is a reduction of the model size. By quantizing down to 4b log including sign bit, we compress the FC weights for free significantly from 1.9 Gb to 0.27 Gb for AlexNet and 4.4 Gb to 0.97 Gb for VGG16. This is because the dense FC layers occupy 98.2% and 89.4% of the total model size for AlexNet and VGG16 respectively.

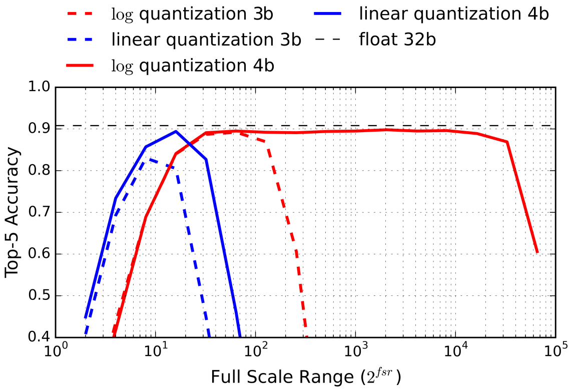

Figure 5. Top5 Accuracy vs Full scale range: VGG16

4.3. Logarithmic Representation of Weights of Convolutional Layers

We now represent the convolutional layers using the same procedure. We keep the representation of activations at 4b log and the representation of weights of FC layers at 4b log, and compare our log method with the linear reference and ideal floating point. We also perform the dot products using two different bases: 2, $\sqrt{2}$. Note that there is no additional overhead for log base-$\sqrt{2}$ as it is computed with the same equation shown in Equation 4.

Table 5 shows the classification results. The results illustrate an approximate 6% drop in performance from floating point down to 5b base-2 but a relatively minor 1.7% drop for 5b base-$\sqrt{2}$. They includes sign bit. There are also some important observations here.

Table 5. Top-5 accuracy after applying quantization to weights of convolutional layers

Model	Float 32b	Linear 5b	Base-2 Log 5b	Base-$\sqrt{2}$ Log 5b
AlexNet	76.8%	73.6%	70.6%	75.1%
VGG16	89.5%	85.1%	83.4%	89.0%

We first observe that the weights of the convolutional layers for AlexNet and VGG16 are more sensitive to quantization than are FC weights. Each FC weight is used only once per image (batch size of 1) whereas convolutional weights are reused many times across the layer's input activation map. Because of this, the quantization error of each weight now influences the dot products across the entire activation volume. Second, we observe that by moving from 5b base-2 to a finer granularity such as 5b base-$\sqrt{2}$, we allow the network to 1) be robust to quantization errors and degradation in classification performance and 2) retain the practical features of $\log$-domain arithmetic.

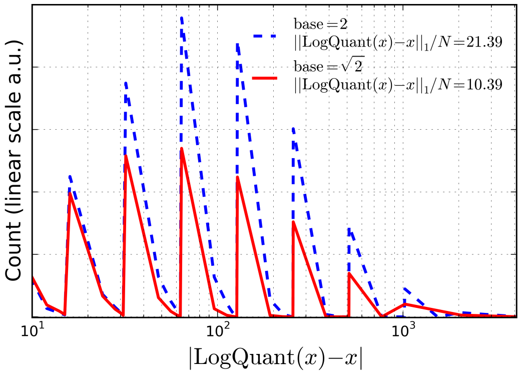

Figure 6. Distribution of quantization errors for weights under base 2 and $\sqrt{2}$.

The distributions of quantization errors for both 5b base-2 and 5b base-$\sqrt{2}$ are shown in Figure 6. The total quantization error on the weights, $\frac{1}{N}||\mathrm{Quantize}(x)-x||_1$, where x is the vectorized weights of size N, is $2\times$ smaller for base-$\sqrt{2}$ than for base-2.

4.4. Training with Logarithmic Representation

We incorporate $\log$ representation during the training phase. This entire algorithm can be computed using Method 2 in Section 3.2. Table 6 illustrates the networks that we compare. The proposed $\log$ and linear networks are trained at the same resolution using 4-bit unsigned activations and 5-bit signed weights and gradients using Algorithm 1 on the CIFAR10 dataset with simple data augmentation described in (He et al., 2015). Note that unlike BinaryNet (Courbariaux & Bengio, 2016), we quantize the backpropagated gradients to train $\log$-net. This enables end-to-end training using logarithmic representation at the 5-bit level. For linear quantization however, we found it necessary to keep the gradients in its unquantized floating-point precision form in order to achieve good convergence. Furthermore, we include the training curve for BinaryNet, which uses unquantized gradients.

Fig. 7 illustrates the training results of $\log$, linear, and BinaryNet. Final test accuracies for $\log$-5b, linear-5b, and BinaryNet are 0.9379, 0.9253, 0.8862 respectively where linear-5b and BinaryNet use unquantized gradients. The test results indicate that even with quantized gradients, our proposed network with $\log$ representation still outperforms the others that use unquantized gradients.

Algorithm 1 Training a CNN with base-2 logarithmic representation. C is the softmax loss for each minibatch. LogQuant(x) quantizes x in base-2 $\log$-domain. The optimization step Update(W_k,g_{W_k}) updates the weights W_k based on backpropagated gradients g_{W_k}. We use the SGD with momentum and Adam rule.

```
Require: a minibatch of inputs and targets $(a_0, a^*)$, previous weights $W$.
Ensure: updated weights $W^{t+1}$
  {1. Computing the parameters' gradient:}
  {1.1. Forward propagation:}
  for $k = 1$ to $L$ do
    $W_k^q \leftarrow \mathrm{LogQuant}(W_k)$
    $a_k \leftarrow \mathrm{ReLU}\left(a_{k-1}^q W_k^b\right)$
    $a_k^q \leftarrow \mathrm{LogQuant}(a_k)$
  end for
  {1.2. Backward propagation:}
  Compute $g_{a_L} = \frac{\partial C}{\partial a_L}$ knowing $a_L$ and $a^*$
  for $k = L$ to $1$ do
    $g_{a_k}^q \leftarrow \mathrm{LogQuant}(g_{a_k})$
    $g_{a_{k-1}} \leftarrow g_{a_k}^q W_k^q$
    $g_{W_k} \leftarrow g_{a_k}^{q\top} a_{k-1}^q$
  end for
  {2. Accumulating the parameters' gradient:}
  for $k = 1$ to $L$ do
    $W_k^{t+1} \leftarrow \mathrm{Update}(W_k, g_{W_k})$
  end for
```

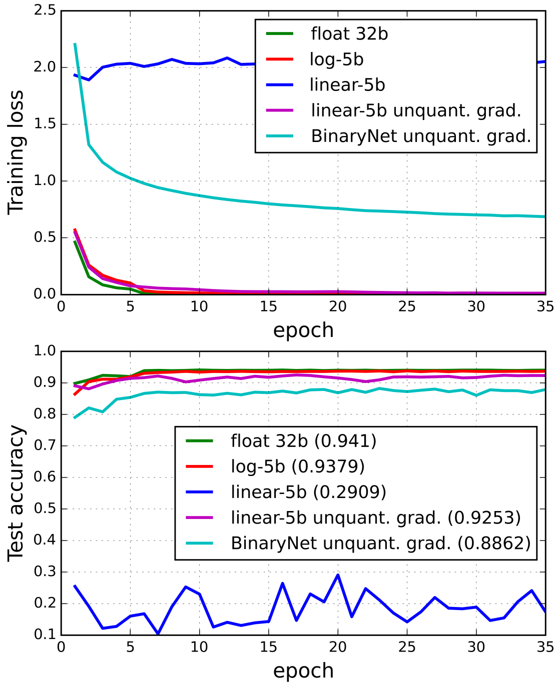

Figure 7. Loss curves and test accuracies

5. Conclusion

In this paper, we describe a method to represent the weights and activations with low resolution in the $\log$-domain, which eliminates bulky digital multipliers. This method is also motivated by the non-uniform distributions of weights and activations, making $\log$ representation more robust to quantization as compared to linear. We evaluate our methods on the classification task of ILSVRC-2012 using pretrained models (AlexNet and VGG16). We also offer extensions that incorporate end-to-end training using $\log$ representation including gradients.

Table 6. Structure of VGG-like network for CIFAR10

log quantization	linear quantization	BinaryNet
Conv $64 \cdot 3 \cdot 3^2$	Conv $64 \cdot 3 \cdot 3^2$	Conv $64 \cdot 3 \cdot 3^2$
BatchNorm	BatchNorm	BatchNorm
ReLU	ReLU	-
LogQuant	**LinearQuant**	**Binarize**
Conv $64 \cdot 64 \cdot 3^2$	Conv $64 \cdot 64 \cdot 3^2$	Conv $64 \cdot 64 \cdot 3^2$
BatchNorm	BatchNorm	BatchNorm
ReLU	ReLU	-
LogQuant	**LinearQuant**	**Binarize**
MaxPool 2×2	MaxPool 2×2	MaxPool 2×2
Conv $128 \cdot 64 \cdot 3^2$	Conv $128 \cdot 64 \cdot 3^2$	Conv $128 \cdot 64 \cdot 3^2$
BatchNorm	BatchNorm	BatchNorm
ReLU	ReLU	-
LogQuant	**LinearQuant**	**Binarize**
Conv $128 \cdot 128 \cdot 3^2$	Conv $128 \cdot 128 \cdot 3^2$	Conv $128 \cdot 128 \cdot 3^2$
BatchNorm	BatchNorm	BatchNorm
ReLU	ReLU	-
LogQuant	**LinearQuant**	**Binarize**
MaxPool 2×2	MaxPool 2×2	MaxPool 2×2
Conv $256 \cdot 128 \cdot 3^2$	Conv $256 \cdot 128 \cdot 3^2$	Conv $256 \cdot 128 \cdot 3^2$
BatchNorm	BatchNorm	BatchNorm
ReLU	ReLU	-
LogQuant	**LinearQuant**	**Binarize**
Conv $256 \cdot 256 \cdot 3^2$	Conv $256 \cdot 256 \cdot 3^2$	Conv $256 \cdot 256 \cdot 3^2$
BatchNorm	BatchNorm	BatchNorm
ReLU	ReLU	-
LogQuant	**LinearQuant**	**Binarize**
Conv $256 \cdot 256 \cdot 3^2$	Conv $256 \cdot 256 \cdot 3^2$	Conv $256 \cdot 256 \cdot 3^2$
BatchNorm	BatchNorm	BatchNorm
ReLU	ReLU	-
LogQuant	**LinearQuant**	**Binarize**
Conv $256 \cdot 256 \cdot 3^2$	Conv $256 \cdot 256 \cdot 3^2$	Conv $256 \cdot 256 \cdot 3^2$
BatchNorm	BatchNorm	BatchNorm
ReLU	ReLU	-
LogQuant	**LinearQuant**	**Binarize**
MaxPool 2×2	MaxPool 2×2	MaxPool 2×2
FC $1024 \cdot 256 \cdot 4^2$	FC $1024 \cdot 256 \cdot 4^2$	FC $1024 \cdot 256 \cdot 4^2$
BatchNorm	BatchNorm	BatchNorm
ReLU	ReLU	-
LogQuant	**LinearQuant**	**Binarize**
FC $1024 \cdot 1024$	FC $1024 \cdot 1024$	FC $1024 \cdot 1024$
BatchNorm	BatchNorm	BatchNorm
ReLU	ReLU	-
LogQuant	**LinearQuant**	**Binarize**
FC $10 \cdot 1024$	FC $10 \cdot 1024$	FC $10 \cdot 1024$
-	-	BatchNorm

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, Ghemawat, Sanjay, Goodfellow, Ian, Harp, Andrew, Irving, Geoffrey, Isard, Michael, Jia, Yangqing, Jozefowicz, Rafal, Kaiser, Lukasz, Kudlur, Manjunath, Levenberg, Josh, Mané, Dan, Monga, Rajat, Moore, Sherry, Murray, Derek, Olah, Chris, Schuster, Mike, Shlens, Jonathon, Steiner, Benoit, Sutskever, Ilya, Talwar, Kunal, Tucker, Paul, Vanhoucke, Vincent, Vasudevan, Vijay, Viégas, Fernanda, Vinyals, Oriol, Warden, Pete, Wattenberg, Martin, Wicke, Martin, Yu, Yuan, and Zheng, Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

Audhkhasi, Kartik, Osoba, Osonde, and Kosko, Bart. Noise benefits in backpropagation and deep bidirectional pre-training. In *Proceedings of The 2013 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2013.

Bishop, Christopher M. Training with noise is equivalent to tikhonov regularization. In *Neural Computation*, pp. 108–116, 1995.

Bottou, Léon and Bousquet, Olivier. The tradeoffs of large scale learning. In Platt, J.C., Koller, D., Singer, Y., and Roweis, S.T. (eds.), *Advances in Neural Information Processing Systems 20*, pp. 161–168. Curran Associates, Inc., 2007.

Chetlur, Sharan, Woolley, Cliff, Vandermersch, Philippe, Cohen, Jonathan, Tran, John, Catanzaro, Bryan, and Shelhamer, Evan. cudnn: Efficient primitives for deep learning. In *Proceedings of Deep Learning and Representation Learning Workshop: NIPS 2014*, 2014.

Courbariaux, Matthieu and Bengio, Yoshua. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Denton, Emily, Zaremba, Wojciech, Bruna, Joan, LeCun, Yann, and Fergus, Rob. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems 27 (NIPS2014)*, pp. 1269–1277, 2014.

Farabet, Clément, Martini, Berin, Akselrod, Polina, Talay, Selçuk, LeCun, Yann, and Culurciello, Eugenio. Hardware accelerated convolutional neural networks for synthetic vision systems. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS),*, pp. 257–260. IEEE, 2010.

Gautschi, Michael, Schaffner, Michael, Gurkaynak, Frank K., and Benini, Luca. A 65nm CMOS 6.4-to-29.2pJ/FLOP at 0.8V shared logarithmic floating point unit for acceleration of nonlinear function kernels in a tightly coupled processor cluster. In *Proceedings of Solid- State Circuits Conference - (ISSCC), 2016 IEEE International*. IEEE, 2016.

Gupta, Suyog, Agrawal, Ankur, Gopalakrishnan, Kailash, and Narayanan, Pritish. Deep learning with limited numerical precision. In *Proceedings of The 32nd International Conference on Machine Learning (ICML2015)*, pp. 1737–1746, 2015.

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

Han, Song, Pool, Jeff, Tran, John, and Dally, William. Learning both weights and connections for efficient neural network. In *Proceedings of Advances in Neural Information Processing Systems 28 (NIPS2015)*, pp. 1135–1143, 2015b.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia*, pp. 675–678. ACM, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C.J.C., Bottou, L., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105, 2012.

Lin, Zhouhan, Courbariaux, Matthieu, Memisevic, Roland, and Bengio, Yoshua. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*, 2015.

Neelakantan, Arvind, Vilnis, Luke, Le, Quoc V., Sutskever, Ilya, Kaiser, Lukasz, and Karol Kurach, James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

Novikov, Alexander, Podoprikhin, Dmitry, Osokin, Anton, and Vetrov, Dmitry. Tensorizing neural networks. In *Advances in Neural Information Processing Systems 28 (NIPS2015)*, pp. 442–450, 2015.

Shin, Sungho, Hwang, Kyuyeon, and Sung, Wonyong. Fixed point performance analysis of recurrent neural networks. In *Proceedings of The 41st IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP2016)*. IEEE, 2016.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:11409.1556*, 2014.

Sung, Wonyong, Shin, Sungho, and Hwang, Kyuyeon. Resiliency of deep neural networks under quantization. *arXiv preprint arXiv:1511.06488*, 2015.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *CVPR 2015*, 2015.

Tokui, Seiya, Oono, Kenta, Hido, Shohei, and Clayton, Justin. Chainer: a next-generation open source framework for deep learning. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)*, 2015.

Vanhoucke, Vincent, Senior, Andrew, and Mao, Mark Z. Improving the speed of neural networks on cpus. In *Proceedings of Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

Zhang, Chen, Li, Peng, Sun, Guangyu, Guan, Yijin, Xiao, Bingjun, and Cong, Jason. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of 23rd International Symposium on Field-Programmable Gate Arrays (FPGA2015)*, 2015.